School of Electrical Engineering and Computing
Department of Electrical and Computer Engineering

Adaptive Second-order Derivative Approximate Greatest Descent Optimization for Deep Learning Neural Networks

Tan Hong Hui

This thesis is presented for the degree of
Doctor of Philosophy
of
Curtin University

October 2019

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature:

Date: 15th October 2019

Thesis Committee

Chairperson: A/Prof. Zhuquan Zang

Supervisor: A/Prof. King Hann Lim

Co-Supervisor: Dr. Raymond Choo Wee Chiong

Acknowledgements

First and foremost, I would like to express my gratitude to my lead supervisor A/Prof. Garenth Lim King Hann for his continuous guidance and encouragement throughout my studies. He literally turned me from nobody to somebody in regards to this field of study. I have learnt not only new knowledge but his dedication, passion and determination that drive me to the completion of my thesis. I appreciate all the time and efforts he spent to improve the quality of my research.

Next, I would like to thank Prof. Goh Bean San, A/Prof. Zang Zhuquan, Dr. Hendra G. Harno and Dr. Raymond Chiong Choo Wee. The valuable comments and suggestions from them had significantly improved the presentation of this thesis. The completion of this thesis would not have been possible without the participation and assistance from them.

Moreover, I would like to thank my research peers for companions throughout my time here. Those positive encouragements are invaluable to keep me moving and stay on track with my research.

To all my relatives and friends who in one way or another shared their support, either morally or financially, I thank you for your generous understanding, encouragement and prayer. I would also like to extend my appreciation to my father and siblings for their supports while I am away from home. To my mother, I thank you for spiritually accompanying me during my tough time, I will never forget what you have taught me.

This research is supported by Malaysia Ministry of Higher Education (MOHE) under the Fundamental Research Grant Scheme (FRGS) with project ID: FRGS/1/2015/TK04/CURTIN/02/1. The support from NVIDIA Corporation is invaluable for donating Titan X GPU in this research. I would also like to thank Curtin University for the financial and administrative support throughout my study.

Abstract

Numerical optimization is essential in artificial neural network backpropagation to minimize an objective function to search for optimal weight. The current optimization framework suffers from the weight initialization, hyperparameter fine tuning and local minimal trap with vanishing gradient issue. Approximate Greatest Descent (AGD) emerges as a new numerical optimization framework that incorporates long-term optimal control theory. This technique computes the second-order derivative Hessian to obtain adaptive step-length, which enables two-phase switching strategy to minimize the objective function of neural networks. In phase-I, multiple spherical search regions are constructed to look for optimal step-length (radius) moving towards the boundary of level set. When the optimizer hovers closer to the region of solution, the optimizer is automatically switched to approximate Newton method in phase-II. Stochastic Diagonal AGD (SDAGD) is proposed in this research work to realize the implementation of AGD in neural networks. Two Hessian approximations are applied into SDAGD, i.e. (a) dropping off-diagonal terms of Hessian with respect to weights and, (b) applying truncated Hessian approximation to remove the higher-order differential terms. The convergence analysis of SDAGD is proven using Lyapunov stability theorem. In the experiment, a two-layer shallow multilayer perceptron (MLP) is built to test on the SGD, SDLM and SDAGD optimizers using MNIST dataset. Due to the adaptive learning using Hessian-based method, the steep descent in the learning weights and errors of SDAGD can be observed during the training process. The proposed SDAGD achieves misclassification rate of 3.34% as compared to SGD (3.81%) and SDLM (4.00%). The experiment also defines the radial effect of SDAGD annealing from 1 to 0.01 to provide better performance of 1.62%. By annealing the radius of the spherical search regions, SDAGD performs normalization to scale

up the relative step-length looking for optimal solution at the optimization level set of phase-II.

To further validate the optimization algorithms, the optimization strategy is visualized using three-dimensional error surfaces by applying SDAGD to solve three topographies of commonly seen error surfaces i.e.: (a) a hilly error surface with two local minima and one global minimum; (b) a deep Gaussian trench to simulate drastic gradient changes experienced with ravine topography and (c) small initial gradient to simulate a plateau terrain. As a result, the long-term optimal control of SDAGD possesses the capability to converge at the fastest rate to the global minimum for problem (a), while other optimizers such as Gradient Descent (GD), AdaGrad and AdaDelta converge towards the local minima. In problem (b) and (c), SDAGD demonstrates the adaptive learning rate element with the second-order derivative information, which is capable to deal with ravine and plateau topographies and converge to the expected solution. On the other hand, gradient-based learning method always encounters the issue of vanishing gradient in neural network backpropagation, causing the weight training process halted without reaching to the final solution. Current practice to overcome this issue is to increase the variants of neural network architecture by using the unsaturated activation function such as Rectified Linear Unit (ReLU). Unsaturated activation function may lead to exploding gradient issue if the hyperparameters are not properly tuned. In the experiments, MLP structure sequentially adding layer by layer is tested with the proposed SDAGD to study the effects of vanishing gradient using saturated and unsaturated activation functions. Deep feedforward neural network up to five hidden layers is evaluated with SDAGD and SGD. The results show that SDAGD is able to obtain good performance in the deep feedforward network; while SGD obtain the worst misclassification error from three to five hidden layers. This result concludes that SDAGD can mitigate the vanishing gradient by avoiding error backpropagation in smaller gradient due to the adaptive learning rate element. SDAGD achieves significantly lower misclassification rate of 2.34% as compared to properly tuned SGD at 9.22% using MNIST dataset.

To evaluate the capability with large-scale optimization problem, SDAGD is applied to deep convolutional neural network by using large-scale dataset

as test cases. DNN refers to a complex neural network architecture, where many layers of information processing stages arranged in a hierarchical structure are trained for pattern classification. DNN is often utilized in image classification in computer vision and image classification, due to self-learning capabilities of important features from convolutional neural network (CNN). Examples of CNN comprises of LeNet-5, AlexNet and ResNet-34. LeNet-5 is the pioneer application of CNN in computer vision while AlexNet outperforms the usage of CNN in image classification using ADAM optimizer. As the datasets are large and complex, increasing depth of CNN architecture has become the practice in the research of machine learning. ResNet-34 improves standard deep CNN architecture by adding residual blocks into the forward propagating structure to solve the vanishing gradient issue. In the experiment of optimization validation, CIFAR-10 and CIFAR-100 datasets of more than 50,000 training images with 10 and 100 output classes respectively are tested with AlexNet and ResNet-34. From the experiment of AlexNet with CIFAR-10 dataset, SDAGD algorithm achieves a misclassification rate of 13.8% which is comparable to ADAM algorithm with 13.6%. As for large-scale image classification task, SDAGD is utilized in ResNet-34 with CIFAR-10 and CIFAR-100 dataset. SDAGD with annealing radius on ResNet-34 with CIFAR-10 dataset achieves 7.98% of misclassification rate when compared to ADAM at 11.11%. As for ResNet-34 with CIFAR-100 dataset, SDAGD algorithm achieves misclassification rate of 33.62%, which is lower than ADAM optimizer at 35.67%. As a conclusion, SDAGD with annealing radius algorithm is able to provide consistent steeper training curve than other methods with higher recognition rate by using the adaptive learning rate derived based on long-term optimal control theory.

Publications

Parts of this thesis and concepts from it have been previously published in the following journal or conference papers.

Journal Papers

1. H. H. Tan and K. H. Lim, "Two-phase Switching Optimization Strategy in Deep Neural Networks", submitted for review.
2. H. H. Tan and K. H. Lim, "Vanishing Gradient Analysis in Stochastic Diagonal Approximate Greatest Descent Optimization", *Journal of Information Science and Engineering*.
3. K. H. Lim and H. H. Tan, "Approximate Greatest Descent in Neural Networks Optimization", *Numerical Algebra, Control & Optimization*, vol. 8, no.3, pp. 337-346.

Conference Papers

1. H. H. Tan and K. H. Lim, "Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization", in *7th International Conference on Smart Computing and Communications (ICSCC)*, Miri, Malaysia, June 28-30, 2019, pp.1-4.
2. H. H. Tan and K. H. Lim, "Review of Second-order Optimization Techniques in Artificial Neural Networks Backpropagation", in *IOP Conference Series: Materials Science and Engineering*, vol. 495, pp. 012003.
3. H. H. Tan and K. H. Lim, "Minibatch Approximate Greatest Descent on CIFAR-10 Dataset", in *IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, Kuching, Malaysia, December 3-6, 2018, pp.320-323.

4. H. H. Tan and K. H. Lim and H. G. Harno, "Stochastic Diagonal Approximate Greatest Descent in Convolutional Neural Networks", in *International Conference on Signal and Image Processing Application (ICSIPA)*, Kuching, Malaysia, September 12-14, 2017, pp. 451-454.
5. H. H. Tan and K. H. Lim and H. G. Harno, "Radial Effect in Stochastic Diagonal Approximate Greatest Descent", in *International Conference on Signal and Image Processing Application (ICSIPA)*, Kuching, Malaysia, September 12-14, 2017, pp. 226-229.
6. H. H. Tan and K. H. Lim and H. G. Harno, "Stochastic Diagonal Approximate Greatest Descent in Neural Networks", in *International Joint Conference on Neural Networks (IJCNN)*, Alaska, USA, May 14-19, 2017, pp. 1895-1898.

Attribution Statements:
All authors provide equal contribution to the completion of the publication listed above.

Contents

List of Figures

List of Tables

List of Acronyms

AdaDelta	Adaptive Per-dimension Learning Rate Algorithm
AdaGrad	Adaptive Gradient Algorithm
ADAM	Adaptive Moment Estimation
BFGS	Broyden-Fletcher-Golfarb-Shanno
BP	Backpropagation
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
DNN	Deep Learning Neural Network
GD	Gradient Descent
GN	Gauss-Newton
GPU	Graphical Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LM	Levenberg-Marquardt
LReLU	Leaky ReLU
MCR	Misclassification rate
MLP	Multilayer Perceptron
MNIST	Mixed National Institute of Standard and Technology
MSAF	Multistate Activation Function

MSE	Mean Squared Error
NAG	Nesterov's Accelerated Gradient
NLP	Natural Language Processing
PReLU	Parametric Rectified Linear Unit
QN	Quasi-Newton
ReLU	Rectified Linear Unit
ResNet	Residual Network
RNN	Recurrent Neural Network
SDAGD	Stochastic Diagonal Approximate Greatest Descent
SDLM	Stochastic Diagonal Levenberg-Marquardt
SGD	Stochastic Gradient Descent

Chapter 1

Introduction

Deep learning neural network (DNN) refers to a complex neural network architecture, where many layers of information processing stages arranged in hierarchical architectures are trained for pattern classification with supervised/unsupervised learning [1]. This field of study has an intersection among other research areas of artificial neural networks i.e., graphical modelling, optimization, pattern recognition, and signal processing. The development of DNN is gaining popularity due to three factors [2]: (a) the enhancement of chip's processing capabilities (e.g. graphical processing units (GPUs)), (b) the improved cost effectiveness of computing hardware, and (c) the recent advances in machine learning and signal/information processing research. These advances have allowed deep learning methods to not only manipulate non-linear functions, but also to learn distributed and hierarchical feature representations in both labelled and unlabelled data. Some of the significant neural network architectures include: Multilayer Perceptron [3], Deep Belief Network [4], Convolutional Deep Belief Network [5], AlexNet [6], VGGNet [7], Residual Network [8] and Long Short-term Memory Recurrent Neural Network [9]. Recently, deep learning has also integrated into various fields of interest including image recognition [6, 10, 11], speech recognition [12–14], new drug or pharmaceutical discovery [15], enhancement of autonomous vehicles [16] and superior performance of an autobot designed for GO chess [17]. In additional to the structural advancement of DNN, the regularization and optimization [18] aspect of DNN specifically in terms of feature representation at sequentially higher abstract level also contributed to the accomplishment of DNN. The regularization effect is induced by utilizing unsupervised learning to bias the learning dynamics in order to initialize the weight parameters in the basin of attraction of a proper local minimum (or in terms of generalization error). Alternatively,

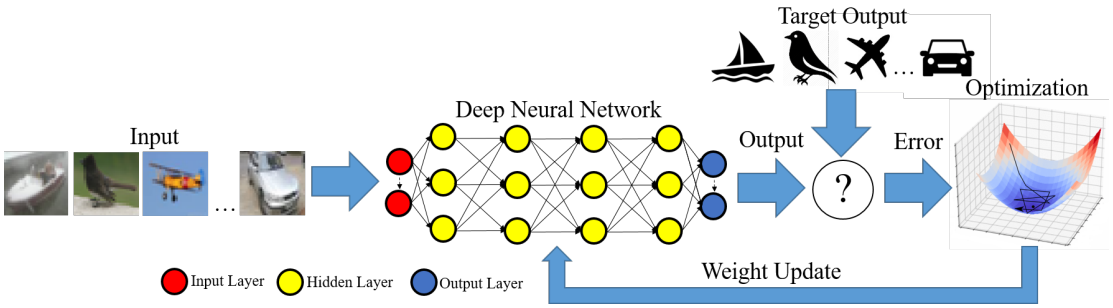

Figure 1.1: General block diagram for optimization in deep learning neural network.

the optimization effect harbors critical challenges because the top two layers of DNN are often over-fitted by the training regardless of the useful features at the lower layers.

Optimization in DNN represents the non-linear learning framework utilized to provide iterative updates to the network for better inference. Artificial neural network contains numerous weight parameters connected in a cascaded layers of network structure. The output of the network computed based on the input is compared with target output to compute the error. Error signal then backpropagates to the network via optimization techniques as shown in Figure 1.1. Conventional optimization techniques are plagued with the problem of vanishing gradient and exploding gradient. As the network grows larger and deeper, the error signal continues to degrade while backpropagating from the output to the input layer [1]. Thus, the optimization will fail early in the training phase and then lead to poor recognition rates. Besides, first-order derivative method utilizing gradient-based information is often trapped into the local minima [19]. In contrast, the second-order derivative method incorporates the Hessian-based local curvature information to avoid bad local minima and are generally require fewer training iterations to achieve optimal solution. In conjunction with the gradient-based approach, conventional approaches usually require hyperparameter fine-tuning as the settings of hyperparameters are not leveraged for general applications [19]. This heuristic hyperparameter tuning contributes to another drawback for neural network training. The evolution leads to an adaptive approach where the learning rate is adjusted automatically according to the state of training. The adaptive learning approaches could also solve the problem with exploding and vanishing gradient on deeper networks. However, the current adaptive approaches are lack of the adaptation of long-term and short-term optimal trajectory optimization mechanism as demonstrated in the approximate greatest descent algorithm [20]. The algorithm proves that having different strategies for

different stages of optimization scenarios is more effective than a standardized approach [21]. In addition, the learning capabilities of deep learning neural networks can be greatly enhanced by utilizing Hessian-based information for better iteration trajectory construction to skip bad local minima.

In this chapter, an overview of the current trends and advancements of deep learning neural network are discussed. The underlying problems of artificial neural networks optimization is also covered briefly as well. Section 1.1 lists down some important research problems in depth while setting a up rigid guidelines and direction for this research. The objectives of this research and significance are discussed in Section 1.2 and 1.3 respectively. Lastly, an overview of the organization and contribution from each chapter in this thesis is summarized in Section 1.4.

1.1 Research Problems

Deep learning neural network is often associated with an efficient optimization technique to look for optimal solution with faster speed of convergence and excellent recognition rate. Although there are several state-of-the-art optimization technique available in the market, each optimizer still possesses its own problem with ample room for improvement. Hence, the research problems are outlined as follows:

- Most commonly adopted state-of-the-art optimization techniques are developed based on gradient-based approach. Gradient-based approach computes only the gradient information to seek for optimal solution by assuming the searching region with a convex model. This assumption has caused the optimizer to be prone to initialization, local minima and vanishing gradient issues. Hence, extensive hyperparameter fine-tuning, such as learning rate, regularization parameters and choice of initialization parameters are required to obtain optimal results [22, 23]. Moreover, the fine-tuned hyperparameter settings are heuristic and will not generalized to all datasets and problems.
- As the datasets in machine learning nowadays are large and complex, increasing depth of DNN architecture has become the common practices. Hence, the training phase of deep learning applications are often associated with extensive training epochs. This is due to the utilization of short-term optimization approach by the current optimization techniques. Existing optimizers tend to compute trajectory based on the error minimization at the present iteration, which requires more

training epoch to arrive at the optimal solution. Long-term optimal trajectory on the other hand, computes trajectory based on the global scale [21, 24] which converges faster to the optimal solution with fewer epochs.

- Poor generalization ability and out-of-sample behavior of adaptive learning rate approach often discourages the usage of adaptive approach in DNN training [25, 26]. Although adaptive learning rate approach alleviates the effort hyper-parameter fine-tuning, the short-term optimization behavior of existing adaptive method suffers by failing to classify new input data properly. Short-term optimization approach is not ideal as it is searching for solution closer to the initial point while the solution is usually far away from the initial point. As a result, a hybrid strategy by utilizing adaptive method at the initial phase of training, followed by non-adaptive method at the last phase of training emerges as one of the viable solutions.

With the concern of solving these issues, the proposed algorithm aims to address the following research questions:

1. How can the proposed method reduce the necessity of fine-tuning hyperparameters while maintaining optimal recognition accuracy?
2. How can the proposed method achieve faster error convergence during training with reduced number of iterations by using long-term optimal control theory?
3. How can the proposed algorithm reduce the error rate and improve the recognition accuracy with two-phase switching optimization strategy?

1.2 Objectives

In this thesis, an adaptive second-order derivative optimization framework derived upon approximate greatest descent will be developed to replace the existing optimization techniques in DNN training. Furthermore, extensive experiments will be conducted to demonstrate the improvements in DNN's learning ability with simple structural changes. The aim of this research is addressed in the following sub-objectives:

1. To design and implement an adaptive second-order derivative optimization framework based upon Approximate Greatest Descent in DNN, by incorporating the computation of Hessian into training phase adaptively for faster convergence trajectory towards the minimum point.

2. To compare the mean squared error and recognition rate of the proposed method with other state-of-the-art techniques using benchmark image-based datasets for both shallow and deep networks.
3. To formulate the theoretical implementation and convergence analysis in DNN through theoretical proof using Lyapunov function theorem.

1.3 Significance and Contributions

DNN is gaining much popularity in recent years via Microsoft Research (since 2009), Google (since 2011), IBM Research (since 2011), Baidu (since 2012), and Facebook (since 2013) due to straightforward system integration and exceptional accuracy. With constant breakthroughs revolving DNN, more adaptation of DNN not limited to the field of engineering are blooming with wide variety of applications. All in all, these achievements are made vastly due to the advancement of optimization techniques in the past decades. This thesis proposed a novel adaptive second-order derivative Hessian-based approach for artificial neural network optimization. The proposed method includes integration of long-term optimal trajectory with two-phase switching approach to improve the recognition rate, learning capability while retaining structural simplicity of DNN optimization. This thesis reports the first implementation of approximate greatest descent (AGD) algorithm in artificial neural network optimization. Therefore, this research is significant due to:

1. The proposed optimization technique is proven convergence by the Lyapunov function theorem demonstrated in Chapter 3. Proven convergence by Lyapunov stability theorem guarantees that the proposed method is robust to small numerical errors. As a result, the experiments concluded that the proposed method outperformed other existing method with shallow network, justified weight convergence with weight parameter plots and is proven robust against wide range of radius settings. The proposed method is also designed vastly similar to gradient-based approach for convenient integration with no major structural changes, which consecutively eases the adaptation into deep learning neural network.
2. The proposed optimization technique utilizes second-order derivative element to approximate the geometry of the error surface. In comparison, the Hessian-based information grants better trajectory towards the minimum point as shown in the optimization visualization in Chapter 4. Moreover, the proposed method

also incorporates two-phase approach through relative step-length to switch the learning rate automatically according to different training phases. The two-phase approach provides larger steps at the beginning of training to help skipping bad local minima and subsequently reduced to approximate Newton method for quadratic convergence. Combination of the strategies help to avoid trapping in plateau and overrunning ravine thus mitigating the vanishing gradient problem in deep learning neural networks.

3. The proposed optimization technique provides long-term optimal control theory to locate the minimum point with multiple spherical search regions. The spherical search regions are regulated with adaptive switching of relative step-length according to training phases to avoid saturation in top hierarchical features and layers. In addition, the computation cost and memory consumption pose significant impact on deep neural network training. The effective computational memory required for the proposed method is negligible because storing of previous gradient data is not required. The proposed method also integrates truncated Hessian to reduce computation cost of inverse Hessian, which is essential for deep learning neural network optimization. Hence, the proposed algorithm is proven feasible to apply in deep neural network with better system performance and accuracy as shown in Chapter 5.

1.4 Thesis Overview

This thesis reports the use of stochastic diagonal approximate greatest descent (SDAGD) in DNN training to maintain structural simplicity whilst improving the model's inference. There are six chapters and the contribution and purpose of each chapter is summarized as follows:

- Chapter 2: *Literature Review in Numerical Optimization*
  This chapter reviews the past and current state-of-the-art optimization techniques applied in the field of DNN. The optimization techniques related to this work are segregated into three categories: gradient-based approach, Hessian-based approach and adaptive learning rate approach. This chapter also pinpoints the problems with the existing optimization techniques followed by the discussion of possible or existing solutions.

- Chapter 3: *Approximate Greatest Descent in Shallow Neural Networks*
  This chapter provides an insight on the adaptation of approximate greatest descent into neural networks optimization. This chapter also demonstrates the analysis on stability and convergence proof of SDAGD algorithm. The numerical results of shallow neural networks with the proposed method are evaluated to support the theoretical claims.
- Chapter 4: *Vanishing Gradient Mitigation Analysis in Deep Learning Neural Networks*
  This chapter covers the case study of vanishing gradient problem with DNN for the rationale behind potential improvements. The experiment involves implementing the proposed method into deep feedforward network with sigmoid activation function. The results show no sign of vanishing gradient with SDAGD algorithm whilst achieving better results from all configurations.
- Chapter 5: *Deep Convolutional Neural Networks using Stochastic Diagonal Approximate Greatest Descent*
  This chapter provides an extensive step to incorporate the proposed method into deep convolutional neural networks such as LeNet-5, AlexNet and ResNet-34. The experiments involve testing the proposed method with deep convolutional neural networks to examine the validity and reliability of the proposed method in image classification task. The results encompassed testing larger dataset with deep residual network. As a result, the proposed method shows significant improvement in training curve and misclassification rate when compared with other existing techniques.
- Chapter 6: *Conclusion and Future Work*
  This chapter summarizes the findings of this research. Some insights on possible future works and expansion on this topic are also discussed in this chapter.

Chapter 2

Literature Review in Numerical Optimization

2.1 Introduction

Numerical optimization is the minimization or maximization of a function subjected to constraint on the variables [27]. In the context of artificial neural network, optimization refers to the minimization of the network's objective function searching for an optimal weight solution W^* to achieve the lowest error rate. The input data forward propagates through the artificial neural network and compares with the targeting output to compute errors. Subsequently, an objective function is constructed in high dimensional spaces for weight optimization. Current numerical optimizations are usually derived from two general techniques, i.e. line search and trust region. Some examples resemble line search approaches include gradient descent (GD) and Newton method whereas trust region encompasses the Levenberg-Marquardt (LM) method. New variants of combining both line and trust region are introduced to improve the efficiency and robustness of optimization techniques.

In the literature review, optimization can be generally classified into three categories, i.e.gradient-based approach, Hessian-based approach and adaptive learning rate approach. Gradient-based approach computes only gradient information to seek for optimal solution by assuming the searching region is always convex and it is closer to the solution. The assumption has caused the optimization to be prone to initialization, local minima and vanishing gradient issues. Many variants of gradient-based approach

are designed to overcome these issues. However, extensive hyperparameter fine-tuning is often required to achieve optimal performance. On the other hand, the Hessian-based approach computes Hessian based on the local curvature information of the objective function to speed up the searching process. The complexity of Hessian computation and Hessian inversion causes the instability and singularity of optimization. Adaptive learning rate approach is one of the variants developed from gradient-based approach to alleviate the process of hyperparameter fine-tuning. Moments and decaying factors are added to adjust the learning rate iteratively during the training process aiming to improve performance. However, it fails to generalize the hyperparameter to all datasets and problems using the short-term fine-tuning approach. As a consequence, having the assumption of searching for solution closer to the initial point shall be eliminated when designing the optimization algorithm. In addition, the performance of Hessian-based approach, considering the additional local curvature information of that objective function, is much better than the gradient-based approach. With the additional information, the effective training iteration is reduced with faster convergence and lead to lesser tuning of hyperparameters. The improved memory allocation and computational efficiency of modern computer further motivate machine learning practitioners to revisit the benefit of utilizing Hessian-based approach.

This chapter covers the literature review of optimization techniques involved in neural network training. Section 2.2 presents the theory for forward and backpropagation of artificial neural networks. The review is segregated into three categories: the gradient-based approach, Hessian-based approach and adaptive learning rate approach covered in Section 2.3, 2.4 and 2.5 respectively. This chapter also summarizes the feasibility and performance of optimization techniques in deep neural network training with Section 2.6. Comments and suggestions for each optimization techniques in artificial neural network training in term of advantages and limitations are covered in Section 2.7.

2.2 Backpropagation in Artificial Neural Network

Backpropagation (BP) is the fundamental development in artificial neural network to provide the capability of self-learning based on the observing data. Due to the advancement in computing technology and deep learning initiatives [2], the learning in deep architecture of neural networks becomes feasible with the aid of BP theory. BP performs parameters update iteratively by using a smaller set of labelled data to adjust

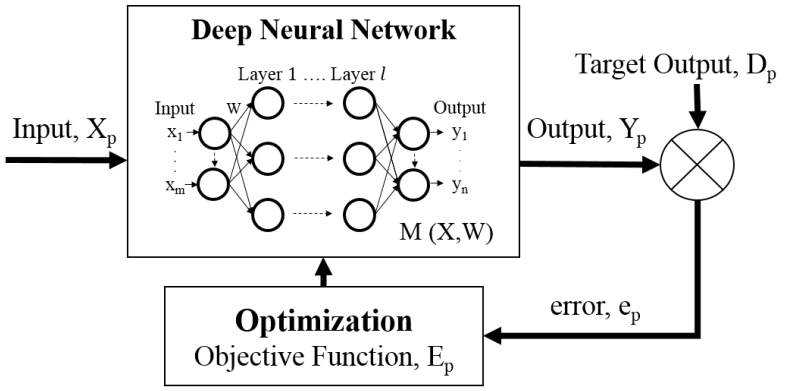

Figure 2.1: General block diagram in deep neural network optimization using backpropagation manner.

the weights between neuron layer by layer from output to input in a backward manner. Standard BP does not perform well in DNN [28, 29] due to the pervasive presence of local optima and other optimization challenges in the non-convex objective function. This problem tends to drive the top hidden layers into saturation easily and becomes more severe as the depth of network architecture increases. Hence, there are manifold of optimization techniques proposed during the past decade to cater these drawbacks.

A generic DNN structure with the BP learning scheme is presented in Figure 2.1. Deep neural network compute a function $Y_p = M(X_p, W)$ where X_p is the p-th input/feature, and W represents the trainable weights in the system. An objective function $E_p = C(D_p, M(X_p, W))$, where $C(\cdot)$ is the cost function constructed in search for optimum minimum point. The cost function computes the difference between targeted output D_p and the output from the system. In practice, Mean Squared Error (MSE) is commonly used as the objective function for DNN and formulated as follows,

$$E(W) = \frac{1}{2P} \sum_{p=1}^{P} \big(D_p - M(X_p, W)\big)^2, \tag{2.1}$$

where P is the total number of inputs. The objective function E is the average sum of errors over a set of input/output training set $\{(X_1, D_1), \cdots (X_p, D_p)\}$. Then, the error signal is backpropagated through the network to modify the optimum value of W that minimizes $E(W)$. This process is called neural network optimization [19]. A full DNN representation can be expressed in multiple layers of transformation as $M(X_p, W) = F^{(l)}(W^{(l)} \times F^{(l-1)}(W^{(l-1)} \times \cdots (F^{(1)}(X_p \times W^{(1)}))))$. For simplicity, the

stacking modules or layers of DNN can be written as,

$$X^{(l)} = F(Y^{(l)}) = F(W^{(l)}X^{(l-1)}), \tag{2.2}$$

where $X^{(l)}$ is the vector output of l-th layer, $W^{(l)}$ is the trainable weights, $X^{(l-1)}$ is the vector input, $Y^{(l)}$ is the total weighted sums to layer l and $F(\cdot)$ is the activation function, e.g., sigmoid activation function. In order to compute the optimal minimal point, chain rule is applied to (2.2) through backpropagation and can be written as,

$$\frac{\partial E}{\partial W^{(l)}} = \frac{\partial E}{\partial Y^{(l)}}\frac{\partial Y^{(l)}}{\partial W^{(l)}} = \frac{\partial E}{\partial Y^{(l)}}X^{(l-1)}, \tag{2.3}$$

$$\frac{\partial E}{\partial Y^{(l)}} = F'(Y^{(l)})\frac{\partial E}{\partial X^{(l)}}, \tag{2.4}$$

where $F'(\cdot)$ is the differentiation of function $F(\cdot)$ with respect to $Y^{(l)}$. The weight update process is computed iteratively using (2.3) to find an optimal solution which minimizes the objective function. Let $\frac{\partial E}{\partial W}$ be the gradient, g_k, the update equation is written as,

$$W_{k+1} = W_k + \eta g_k, \tag{2.5}$$

where W_{k+1} is adjusted from W_k and η is the learning rate parameter. The iteration steps from (2.5) formulated from backpropagation is further generalized into the standard numerical optimization and written as,

$$W_{k+1} = W_k + \alpha_k p_k, \tag{2.6}$$

where learning rate, η is equivalent to the step-length, α_k and the gradient, g_k represents the search direction, p_k. Therefore, numerical optimization techniques play an important role in artificial neural network optimization.

2.3 Gradient-based Approach

The gradient-based approach is effective and computationally efficient against deep and large artificial neural networks. The gradient-based approach utilizes the first-order derivative information to determine the step direction at every training iteration. The vanilla version of gradient-based approach is slow and susceptible to local minima thus always incorporated with the momentum method to accelerate through the training

phase. The performance of momentum method is also bounded by the parameter initialization. Hence, the process involves sequencing through a series of initialization for better performance in terms of accuracy and computational efficiency [30]. The following subsection will cover some gradient-based approaches in DNN training.

2.3.1 Gradient Descent Method

Gradient descent (GD) [27], also known as steepest descent, is one of the most adopted optimization techniques available in the literature. Gradient descent can be evaluated as the negative of the gradient of the objective function E where the descent direction $p_k = -g_k$. The negative sign resembles the descent direction towards the minimum point of an objective function. By integrating gradient descent into the iterative equation yields,

$$w_{k+1} = w_k - \eta g_k. \tag{2.7}$$

This search direction p_k induces a monotonic descent direction towards the minima of the objective function, while η controls the speed of descent. Since there are no local curvature information involved in the equation, gradient descent is prone to be trapped in bad local minima. In order to overcome local minima issues, a variant of GD, stochastic gradient descent (SGD) [31] was proposed. SGD induces stochasticity to the input data by randomly initializing the starting point in a distribution to solve poor local minima issues. The random input sequence of SGD allows skipping through bad mimima and thus providing better overall system performance. Nevertheless, ordinary SGD method creates improper load balancing that requires additional efforts in fine-tuning the hyperparameters [31].

2.3.2 Momentum Method

The momentum method [32] is inspired by the momentum of a snow ball rolling down the hill. In numerical optimization context, the loss of an objective function is the representation of the height of a hilly terrain, simulating a particle rolling down with the help of accumulated potential energy. As a result, momentum method uses accumulated gradient information to update the weights in direction that have persistent reduction in the cost function. This parameter vector help builds up velocity in the direction with consistent gradient to speed up the convergence process. Hence, the update of

momentum method is written as follows,

$$\gamma_{k+1} = \beta\gamma_k - \eta g_k, \tag{2.8}$$

$$w_{k+1} = w_k + \gamma_{k+1}, \tag{2.9}$$

where γ_{k+1} is the accumulated gradients, $\beta \in [0, 1]$ is the momentum decay coefficient. When β is closer to 1, more accumulated gradients are utilized across more iterations which lead to higher velocity. The momentum decay coefficient works like the friction to the slope of the objective function. The momentum decay coefficient damps the effect of accumulated gradients by partially discarding the old gradients. Decay coefficient is a tunable hyperparameter and is essential to avoid large value of gradient information persisting across multiple iterations while maintaining suitable past gradients to skip through the bad local minima. Momentum method performed great in the "low-curvature" regions of objective function by accelerating through low and persistent gradient information. However, momentum method in the "high-curvature" error surface suffers from deceleration. Consequently, momentum method is prone to skip important minima throughout the optimization process.

2.3.3 Nesterov's Accelerated Gradient Method

Nesterov's Accelerated Gradient (NAG) method [33] is a variant to momentum method. NAG provides stronger theoretical convergence guarantees compared to standard momentum method as it computes the gradient after adding the previous accumulated gradients/velocities. Thus, NAG method evaluates gradient at the next iteration as,

$$\gamma_{k+1} = \beta\gamma_k - \eta g_k, \tag{2.10}$$

$$w_{k+1} = w_k - \beta\gamma_k + (1 + \beta)\gamma_{k+1}. \tag{2.11}$$

Similar to momentum method, the decay coefficient β also requires to be tuned but is more sensitive in the case of NAG method. The method requires a schedule of learning rate and momentum decay coefficient for optimum performances. Consequently, NAG changes the velocities faster and is more responsive to the stages of training. Hence, each γ is slightly more effective than vanilla momentum in correcting large and inappropriate velocity resulting in a better recognition performance. While dealing with low-curvature error surfaces, NAG is also experimentally proven better to cancel off the high frequency oscillation that causes ordinary gradient descent to diverge.

2.3.4 Conjugate Gradient Method

Conjugate gradient [34] computes the descent direction with conjugate gradient coefficient. The conjugate gradient coefficient B_k is computed with either Polak and Ribiere [35] method,

$$B_k = \frac{(g_k - g_{k-1})^T g_k}{(g_{k-1})^T g_{k-1}}; \tag{2.12}$$

or the Fletcher and Reeves [36] method as,

$$B_k = \frac{{g_k}^T g_k}{(g_{k-1})^T g_{k-1}}. \tag{2.13}$$

By incorporating the conjugate direction into the update rule, it yields,

$$w_{k+1} = w_k + \eta(-g_k + B_k p_{k-1}), \tag{2.14}$$

where $p_{k-1} = -g_{k-1} + B_{k-1} p_{k-2}$. The usage of conjugate direction reduces the needs of high computation capability for Hessian approximation as it is a $\mathcal{O}(N)$ method. This method does not use the actual Hessian directly, but manipulating the current update with the update direction from previous iteration. On top of that, conjugate gradient uses a line search method to identify the step-length and thus it is only suitable for batch learning. Nevertheless, conjugate gradient descent performs poorly in determining a stopping criteria and requires proper parameter tuning prior to training for optimal results.

2.3.5 Summary of Gradient-based Approach

Gradient-based approach requires minimal computation power and memory due to the involvement of gradient information only. However, there are still multiple drawbacks tied to the original gradient descent method. The conjugate gradient method applies line-search approach for step-length approximation to reduce computation cost but this approach only works with batch training. Subsequently, since gradient descent is essentially designed for convex problem, gradient-based approach often suffers from trapping in bad local minima. The convergence rate is slower because the descent rate is controlled by the step-length only. For instance, the momentum and NAG is essential to provide variable descent rate and the ability to hover through bad local minima. Nevertheless, the added parameters demand for hyperparameter fine-tuning turn out to be slower than Hessian-based approach. Gradient-based approach is also sensitive to parameters initialization as well as involving other regularization methods for better recognition which contributed to more hyperparameters fine-tuning.

2.4 Hessian-based Approach

The Hessian-based approach utilizes the local curvature information of an error surface for optimization, where the weight parameter is expressed as W_k in matrix form. The Hessian information allows optimization to accelerate through the "low-curvature" surface or a plateau, and decelerate when hovering along the "high-curvature" surfaces like a ravine. Hence, the Hessian-based approach usually achieves faster convergence than the gradient-based approach. However, the Hessian-based approach receives less interest in neural network training due to expensive computation of inverse Hessian. The following subsection covers some well known Hessian-based approaches in DNN training.

2.4.1 Newton Method

Newton method is one of the profound numerical optimization techniques created by Isaac Newton and Joseph Raphson [27]. This method iteratively computes the inverse Hessian of an arbitrary point along a quadratic curve until the point converges at the optimum. Newton method is invariant to linear transformations of the input vector due to the quadratic behavior. Hence, Newton method convergence will not be affected by shifts, scaling and rotation of input vectors. Let Hessian $H_k = \frac{\partial^2 E}{\partial W^2}$, the Newton method is written as,

$$W_{k+1} = W_k - \eta H_k^{-1} g_k, \tag{2.15}$$

where H_k^{-1} is the inverse Hessian of the quadratic curve, η must be chosen in the range of $0 < \eta < 1$ since the objective function in practice is not perfectly quadratic. Ideally, convergence usually happens in one step via Newton method if the objective function is quadratic, thus Newton method is particularly useful when W is near to the solution W^*. However, convergence may not be ideal in optimizing neural network's objective function since the objective function is often non-quadratic. Additionally, the Hessian matrix needs to be positive definite to compute inverse Hessian. Lastly, the computational power needed to compute inverse Hessian with large amount of variables is extremely high. Hence, the ordinary form of Newton method is not suitable for artificial neural network training.

2.4.2 Quasi-Newton Method

Quasi-Newton (QN) [37] uses inverse Hessian estimation to compute each update iteration. There are a few estimation algorithms available but the most well-known is the Broyden-Fletcher-Golfarb-Shanno (BFGS) [19] algorithm. The estimated inverse Hessian, Q_k for BFGS is computed at each iteration with the following equation,

$$Q_k = \Big(I - \frac{\delta\phi^T}{\phi^T\delta}\Big)Q_{k-1}\Big(I - \frac{\phi\delta^T}{\phi^T\delta}\Big) + \frac{\delta\delta^T}{\phi^T\delta}, \tag{2.16}$$

where $\delta = W_k - W_{k-1}$ and $\phi = g_k - g_{k-1}$. By substituting the estimated inverse Hessian into the update equation yields,

$$W_{k+1} = W_k - \eta Q_k g_k. \tag{2.17}$$

Quasi-Newton indeed reduces majority of the computation cost with just $\mathcal{O}(N^2)$ computation compared to Newton method. However, Quasi-Newton still suffers from the need of storing $N \times N$ matrices. There is a limited memory version of Quasi-Newton method dubbed as L-BFGS [38] developed with the aim of reducing the required memory space for each iteration. Nevertheless, the reduced memory space is insignificant compared to the gradient-based approach when training large-scale dataset with DNN.

2.4.3 Gauss-Newton Method

Gauss-Newton (GN) [39] method is another alternative to approximate inverse Hessian. Gauss-Newton uses only the squared Jacobian and ignores the second-term of H_k entirely. Let the squared Jacobian matrix, or also known as truncated Hessian be $\bar{H} = {g_k}^T g_k$, the update equation is now written as,

$$W_{k+1} = W_k - \eta \bar{H}^{-1} g_k. \tag{2.18}$$

Gauss-Newton has a complexity of $\mathcal{O}(N^3)$ and is mainly designed for batch learning. However, if the error is large, the truncated Hessian is no longer a valid approximation of full Hessian and thus may cause divergence to happen. Subsequently, if the truncated Hessian is singular and not positive definite, the model will be blown and the iteration steps will no longer go on a descent direction.

2.4.4 Levenberg-Marquardt Method

The Levenberg-Marquardt (LM) [40] method was proposed in response to the drawback lies with the Gauss-Newton method. The method introduces a regularization parameter μ with $\mu > 0$ to contain the problem with indefinite Hessian. The Levenberg-Marquardt method is written as follows,

$$W_{k+1} = W_k - \eta\Big(\bar{H} + \mu I\Big)^{-1} g_k, \tag{2.19}$$

where I is the identity matrix. The effect of regularization parameter μ possesses three significant impacts, i.e.: (a) for a sufficiently large μ, the truncated Hessian is always positive definite to ensure the iteration steps in descent direction. (b) For small μ, the LM method works exactly like a Gauss-Newton method by having quadratic convergence in the final stage. (c) For an overly large μ, the truncated Hessian is ignored, the iteration steps resemble gradient descent method which provides better step at the start of training. The LM method guarantees convergence better than other predecessor methods even if the initial point is far away from the optimal point. Nevertheless, LM method is still an ad-hoc approach by incorporating the regularization parameter without proper theoretical proof and properly documented rationale.

2.4.5 Hessian-free Method

Hessian-free [41] method, also known as truncated-Newton method, works by first computing a scaled-down version of Hessian to determine the local curvature and later implement the conjugate gradient method to optimize further. With similar intuition of reducing the computation of full Hessian, Hessian-free method utilizes a scaled down version of Hessian matrix $\widehat{H}$ with finite differences at the cost of a single extra gradient evaluation via,

$$\widehat{H} = \lim_{\epsilon \to 0} \frac{(g_k + \epsilon) - g_k}{\epsilon}. \tag{2.20}$$

The Hessian approximation applied by Hessian-free method is ingenious as it requires only the matrix-vector products in optimizing the quadratic objectives function. However, Hessian-free works only with the help of conjugate gradient. The behavior of conjugate gradient will naturally make significant progress in minimization along with the training iterations.

2.4.6 Summary of Hessian-based Approach

Hessian-based approach provides faster convergence due to the adaptation of Hessian information for trajectory construction. The biggest downside is high computing cost and the needs to store large Hessian matrix for computation as seen in Newton method. Hence, methods for Hessian approximation or estimation emerge as the alternative for Hessian-based optimization. Firstly, Quasi-Newton solves the computing problem with approximated Hessian. However, BFGS method requires Hessian information from previous iteration which is still prone to memory issues. Hessian-free method complements the memory issue slightly by down-scaling the Hessian matrix followed by conjugate gradient method. Gauss-Newton excludes the needs of storing Hessian information from previous iteration but prone to exploding gradient if the matrix is not positive definite. Lastly, LM method implements the regularization parameter through an ad-hoc manner to contain the indefinite Hessian problem. The fact that the choice of regularization is not properly theorized, LM method still requires hyperparameter fine-tuning as the choice of the regularization parameter is not generalized to all problems.

2.5 Adaptive Learning Rate Approach

Due to the complexity of DNN's problem, a fix selection of learning rate or simply annealing the learning rate will not provide an optimum result. Many efforts and time are invested in hyperparameter fine-tuning and are not leveraged across different problems. Hence, an adaptive approach is essential to have the optimizer tuning the hyperparameters automatically according to training phases. An adaptive method should behave properly to a broader range of hyperparameter settings. The following subsection covers some well-known adaptive approaches in DNN training.

2.5.1 Adaptive Gradient Algorithm (AdaGrad)

Adaptive gradient algorithm (AdaGrad) [23] is an adaptive learning rate optimization algorithm proposed by Duchi et al. to overcome the needs of extensive hyperparameter fine-tuning with other algorithms. It performs an informative gradient-based learning derived from the geometry of data of the earlier iterations. AdaGrad possesses the capability to induce larger updates to infrequent training input while having smaller update for frequent training input. AdaGrad computes the learning rate at every iteration instead of updating the parameters using a global learning rate. AdaGrad's

formula is written as,

$$w_{k+1} = w_k - \eta[\sqrt{(s_k)^2} + \mu]^{-1} g_k, \tag{2.21}$$

where s_k is the cumulative gradient where $s_k = s_{k-1} + g_{k-1}^2$, g_{k-1}^2 is the squared of gradient information and μ is the smoothing/regularization term to avoid division by zero. In consequence, AdaGrad will change the learning adaptively based on the gradient from previous iteration. Nevertheless, the nature of AdaGrad method causes monotonic reduction of learning rate across iteration when s_k is getting larger concurrently. The optimization with AdaGrad will stop early and require longer training time compared to other existing methods.

2.5.2 Adaptive Per-dimension Learning Rate Algorithm (AdaDelta)

Adaptive learning rate algorithm (AdaDelta) [42] is a per-dimension learning rate method for gradient descent algorithm. AdaDelta is designed to overcome the cons with AdaGrad algorithm that has a continuous decaying learning rate. AdaDelta is built based on two main concepts (a) unlike AdaGrad, the cumulative gradient is scaled in a decaying manner where lesser historical gradients are used as the iteration grows, (b) applying momentum based acceleration term to keep the optimization running. As a result, AdaDelta is similar to AdaGrad but without the needs to do manual hyperparameter fine-tuning and is robust to wide range of applications. The update rule of AdaDelta is written as follows,

$$w_{k+1} = w_k - \sqrt{A[\Delta W_k^2] + \mu}\Big[\sqrt{A[g_k^2]} + \mu\Big]^{-1} g_k, \tag{2.22}$$

where $A[g_k^2] = \rho A[g_{k-1}^2] + (1-\rho) g_k^2$ is the decaying accumulated gradient, $A[\Delta w_k^2] = \rho A[\Delta w_{k-1}^2] + (1-\rho)\Delta w_k^2$ is the accumulated deltas for momentum like approach, ρ is the decaying parameter and μ is the smoothing term since the first delta gradient is equal to zero. Consequently, AdaDelta possesses individual learning rates for each parameter but comes with a global momentum adaptation. The next innovation suggests individual momentum acceleration for each parameter.

2.5.3 Adaptive Moment Estimation (ADAM)

Adaptive moment estimation (ADAM) [43] is proposed to enhance the optimization performance of AdaDelta. ADAM is a combination of per-parameter AdaGrad and momentum adaptation. Unlike AdaDelta, ADAM stores the exponentially decaying

average of past deltas just like the per-parameter momentum acceleration. The update rule of ADAM can be written as,

$$w_{k+1} = w_k - \eta[\sqrt{\hat{v}_k} + \mu]^{-1}\hat{m}_k, \tag{2.23}$$

where the first moment estimate is $m_k = \beta_1 m_{k-1} + (1-\beta_1)g_k$, the second raw moment estimate is $v_k = \beta_2 v_{k-1} + (1-\beta_2)g_k^2$. Subsequently, the biased-corrected first moment estimate is $\hat{m}_k = \frac{m_k}{1-\beta_1}$ and the biased-corrected second raw moment estimate is $\hat{v}_k = \frac{v_k}{1-\beta_2}$. ADAM is the current state-of-the-art adaptive optimization technique available for DNN. This is because ADAM provides the fastest convergence and is more robust to problems like vanishing gradient, vanishing learning rate. ADAM is also able to work with sparse gradient and the magnitudes of parameters are invariant to re-scaling of gradients.

2.5.4 Summary of Adaptive Learning Rate Approach

ADAM being the current state-of-the-art combines the benefits of AdaGrad and momentum approach. ADAM is fast, efficient and used in most DNN problems due to extensive theory backing up with minimal components and hyperparameters added for robustness. Nevertheless, ADAM suffers from poor generalization ability towards out-of-sample dataset. Hence, it leads to hybrid strategy of utilizing adaptive method at the initial phase of training, followed by non-adaptive method at later training phase. Moreover, the current adaptive learning rate approaches work only with the gradient-based optimization technique due to heavy bias on computational efficiency. Theoretically, there are still room for improvements to the state-of-the-art adaptive optimization techniques by utilizing Hessian-based approach. The Hessian-based approach achieves faster convergence due to the adaptation of local curvature information. Although Hessian-based approach is more computationally intensive and memory expensive, there are ample improvements on computing hardware to ameliorate the issue with computational cost.

2.6 Review of Numerical Optimization Methods on Benchmark Datasets

This research covers some benchmark datasets for evaluation, i.e. Mixed National Institute of Standard and Technology (MNIST) [44], Canadian Institute for Advanced

Research (CIFAR) [45] and ImageNet [46] datasets. The benchmark image dataset are used in conjunction with state-of-the-art DNN models to verify the generalizability and robustness of the proposed algorithm. Table 2.1 shows the related works classified based on the optimization techniques used in the literature.

Table 2.1: An overview of test error rate for various dataset.

Dataset	Optimization	Reference	Error rate
MNIST [44]	GD	[5, 47–49]	0.21% - 1.19%
	QN	[37, 50, 51]	0.69% - 0.77%
	Newton	[52]	3.52%
	GN/LM	[40, 41, 53]	1.4% - 2.97%
CIFAR [45]	AdaGrad	[54]	69.7%
	AdaDelta	[55, 56]	8.58% -25.16%
	ADAM	[57–59]	3.47% - 17.2%
ImageNet [46]	AdaGrad	[60]	43.4%
	ADAM	[6, 61]	17% - 55.8%

2.7 Chapter Summary

The learning curve of artificial neural network is one of the well-studied area in machine learning. The main implication from the literature review is to identify the effective and robust optimization technique for artificial neural network training. This chapter covers an extensive review on the current state-of-the-art optimization techniques used in the field of DNN learning. The optimization techniques related to this work are segregated into three categories: gradient-based approach, Hessian-based approach and adaptive learning rate approach. Gradient-based approach is more computationally and memory efficient compared to Hessian-based approach. The gradient-based approach often incorporates momentum method to hover through the error surface faster while avoiding convergence towards bad local minima. More hyperparameters are added and thus extra time is invested in fine-tuning for optimal results. Subsequently, Hessian-based approach provides faster convergence with local curvature information but struggle with computation and memory issue.

The adaptive learning rate approach provides variable learning rate by adding more components and hyperparameters into the equation. An adaptive approach allows the training to behave properly to a broader range of hyperparameter settings. Firstly, adaptive gradient algorithm (AdaGrad) [23] incorporates adaptive learning rate into optimization algorithm to reduce the need for extensive hyperparameter fine-tuning. An informative gradient-based learning is performed based on the geometry data of previous iterations. AdaGrad is capable of providing larger updates to infrequent training input while having smaller updates for frequent training input. Additionally, adaptive per-dimension learning rate algorithm (AdaDelta) [42] is a per-dimension learning rate method for gradient descent algorithm which is designed to overcome the limitation of AdaGrad algorithm which has a continuous decaying learning rate. On top of AdaDelta, Adaptive moment estimation (ADAM) [43] is proposed to enhance the optimization performance of AdaDelta by combining the per-parameter updates of AdaGrad and momentum adaptation. Unlike AdaDelta, ADAM stores the exponentially decaying average of past deltas for the per-parameter momentum acceleration. Nevertheless, the current adaptive learning rate approach work only with the gradient-based method due to the emphasis on computational efficiency. As a result, this research will explore the possibility of applying adaptive second-order derivative approach based on Hessian computation into a high-order non-linear deep learning framework to improve DNN's learning ability, while retaining the structural simplicity of DNN. Long-term optimal trajectory shall be adopted in the optimization algorithm design to seek for solution, which is far away from the initial point.

Chapter 3

Approximate Greatest Descent in Shallow Neural Networks

3.1 Introduction

Adaptive learning rate approach [23, 42, 43] has become the current trend in numerical optimization to improve the recognition rate of neural networks using moment and decaying factor. Adaptive gradient algorithm (AdaGrad) [23] incorporates adaptive learning rate into optimization algorithm to reduce the need for extensive hyperparameter fine-tuning. An informative gradient-based learning is performed based on the geometry data of previous iterations. AdaGrad is capable of providing larger updates to infrequent training input while having smaller updates for frequent training input. Adaptive per-dimension learning rate algorithm (AdaDelta) [42] is a per-dimension learning rate method for gradient descent algorithm which is designed to overcome the limitation of AdaGrad algorithm which has a continuous decaying learning rate. Subsequently, adaptive moment estimation (ADAM) [43] is proposed to enhance the optimization performance of AdaDelta by combining the per-parameter updates of AdaGrad and momentum adaptation. ADAM stores the exponentially decaying average of past deltas for the per-parameter momentum acceleration. However, the adaptive learning approaches using gradient-based information fail to generalize the hyperparameters in various datasets due to the adaptation of short-term fine-tuning approach.

On the other hand, the Hessian-based approach make use of curvature information to adjust the learning rate adaptively. The step-length is adjusted iteratively based on the

Hessian computation from the error surface throughout the training process of artificial neural networks. Due to the current advances of computing power and memory allocation in computer hardware, it enables the development of Hessian-based approach in the deep learning neural network optimization. Approximate Greatest Descent (AGD) [20, 62, 63] emerges as a new Hessian-based optimization framework that incorporates long-term optimal control theory. This chapter covers a novel adaptive Hessian-based approach for artificial neural network optimization by utilizing AGD algorithm. This method is equipped with a two-phase switching spherical optimization strategy. The two-phase approach of the proposed method is derived and inspired by the multistage optimal control system. The utilization of two-phase approach enables more strategic plan to deal with vanishing and exploding gradient problem as the step-length is changed according to the state of training. Besides, the conventional approaches suffer from local minima problem because most DNN's error surfaces are plague with multiple plateaus and ravines. The ad-hoc and heuristic approach in existing optimization techniques often fail to converge to an optimal solution with poor generalization capability. To address the current research problems, the proposed method utilizes the long-term optimal trajectory control theory to construct an adaptive step-length to deal with different stages of optimization problem. One of the component in AGD algorithm, relative step-length in particular, is naturally an adaptive optimization approach which switches the learning rate automatically to search for optimal trajectories towards the minimum point.

This chapter covers the generic optimization problem and the implementation of the proposed method. The theory and derivation of approximate greatest descent (AGD) is covered in Section 3.2. Subsequently, Section 3.3 includes the derivation and approximation for stochastic diagonal AGD (SDAGD) algorithm. Convergence analysis is covered in Section 3.4 and the experiment setup to support the theoretical claims are covered in Section 3.5, 3.6 and 3.7 respectively. Lastly, a recap for this chapter is summarised in Section 3.8.

3.2 Theory of Approximate Greatest Descent

Approximate Greatest Descent (AGD) [21, 62, 63] evolves from dynamical control optimization problem and emerges as a new means of numerical optimization technique. AGD introduces the concept of long-term optimal trajectories by proposing different

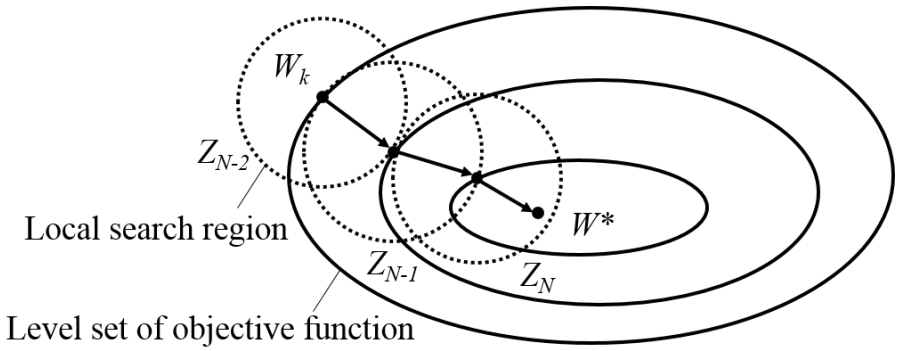

Figure 3.1: Long-term optimal trajectory from initial point to solution of approximate greatest descent algorithm.

strategies at different phases of optimization. A body of research [64–66] had shown that long-term optimal trajectory is the preferred tool to solve an optimization problem through a sequence of closed and finite spherical search regions. If the estimated local minimum does not lie within the spherical search regions, the intermediate solution should be identified at the boundary of the local search regions. Alternatively, when the search region contains the solution, approximate Newton method is used for faster convergence.

Algorithmically, AGD formulates the long-term optimal control approach as a multistage decision control system with a sequence of points at each of the search region. Thus, this leads to a two-phase strategy optimization technique in order to deal with complex and sophisticated optimization problem. The two-phases of AGD method is outlined as, phase-I encompasses the spherical search regions without the minimum point, while phase-II contains the minimum point bounded inside the spherical search region. Phase-I mostly consists of iteration steps that are far away from the solution. AGD iteration generates a sequence of spherical local search regions and has the next iteration step identified at the boundaries. Phase-II is triggered when the iteration step is near to solution, AGD switches strategy to approximate Newton method for quadratic roll-off. Figure 3.1 demonstrates the iteration steps of AGD algorithm simplified in three iterations via long-term optimal trajectory starting from initial point W_k to the solution W^*. AGD is applied in numerical optimization for Rosenbrock and Powell problems [63]. Both problems demonstrated the robustness of AGD optimization techniques with faster optimization, lower computational cost and the ability to handle ill-conditioned and indefinite Hessian.

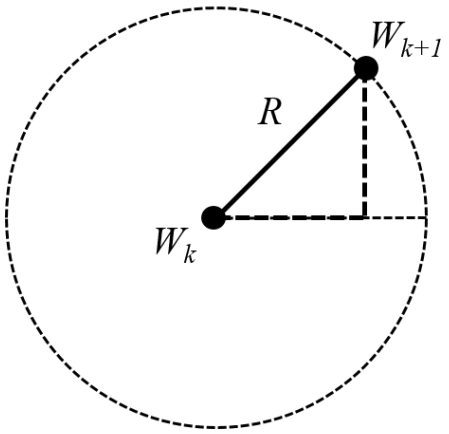

Figure 3.2: AGD iteration with next iteration located at the boundary of the local spherical search region.

To search for minima in an objective function, AGD applies the general iterative equation written as follows,

$$W_{k+1} = W_k + \alpha_k p_k, \tag{3.1}$$

where α_k is step length and p_k is the search direction. AGD formulates the iteration step by using multiple local spherical search regions.

Let the search regions be $Z_1, Z_2, ..., Z_N$ as seen in Figure 3.1, where N represents the total number of local search regions. For every local search region except the last search region Z_N, the local minimum point is located at the boundary of the search region. As for the last local search regions Z_N, AGD approximates Newton method for faster convergence rate when the current point W_k is near to the minimum point W^* within the spherical search regions. Figure 3.2 illustrates one of the AGD iteration identified at the boundary of the search region by minimizing the objective function at the next iteration, $E(W_{k+1})$ subject to:

$$R_k{}^2 = \|W_{k+1} - W_k\|^2, \tag{3.2}$$

where $R_k > 0$ is the radius of the search region. Lagrangian function, $L(W_{k+1})$ is applied to optimize the objective function and can be written as:

$$L(W_{k+1}) = E(W_{k+1}) - \lambda_k[u_k^T u_k - R_k{}^2], \tag{3.3}$$

where $u_k = \alpha_k p_k = W_{k+1} - W_k$ represents the control vector and λ_k represents the Lagrange multiplier. Computing the derivatives of the Lagrangian function with respect

to the control vector $u_k = \alpha_k p_k$ to produce the following equations,

$$\begin{aligned}\frac{\partial L(W_{k+1})}{\partial u_k} &= \frac{\partial E(W_{k+1})}{\partial W_{k+1}} - 2\lambda_k u_k \\ &= \frac{\partial E(W_{k+1})}{\partial W_{k+1}} - 2\lambda_k \alpha_k p_k.\end{aligned} \tag{3.4}$$

To obtain an iterative point on the boundary, the optimal condition will have the derivative of Lagrangian function to be at zero.

$$\frac{\partial E(W_{k+1})}{\partial W_{k+1}} - 2\lambda_k \alpha_k p_k = 0. \tag{3.5}$$

According to [62], the optimal search direction can be achieve in AGD if the Lagrange multiplier is set to $\lambda_k = -\frac{1}{2\alpha_k}$, hence,

$$p_k = -\frac{\partial E(W_{k+1})}{\partial W_{k+1}}. \tag{3.6}$$

However, the gradient of the next iteration step is still unknown at iteration k, thus the optimal search direction in AGD is approximated using current step's gradient,

$$p_k \approx -\frac{\partial E(W_k)}{\partial W_k}. \tag{3.7}$$

For simplicity, recall that $g_k = \frac{\partial E}{\partial W_k}$, $H_k = \frac{\partial^2 E}{\partial W_k^2}$ and apply Taylor series expansion in (3.5) to approximate the next iterative gradient. Hence, the direction vector for the next iteration is written as,

$$\begin{aligned} p_k &= -g_k - \alpha_k H_k p_k \\ p_k I + \alpha_k H_k p_k &= -g_k \\ [I + \alpha_k H_k] p_k &= -g_k \\ p_k &= -[I + \alpha_k H_k]^{-1} g_k, \end{aligned} \tag{3.8}$$

where I is the identity matrix. Subsequently, replacing the direction vector p_k into the general update equation (3.1) produces,

$$\begin{aligned} W_{k+1} &= W_k - \alpha_k [I + \alpha_k H_k]^{-1} g_k \\ &= W_k - [{\alpha_k}^{-1} I + H_k]^{-1} g_k. \end{aligned} \tag{3.9}$$

To solve for step length α_k, the derivative of Lagrange function (3.3) with respect to λ_k

is used. Letting the derivative equation to be at zero for optimality condition, yields,

$$\begin{aligned} \frac{\partial L(W_{k+1})}{\partial \lambda_k} &= {u_k}^T u_k - {R_k}^2 = 0 \\ \alpha_k^2 {p_k}^T p_k &= {R_k}^2 \\ \alpha_k^2 = \frac{R^2}{{p_k}^T p_k} &= \frac{{R_k}^2}{\|g_k\|^2} \\ \alpha_k &= \frac{R_k}{\|g_k\|}. \end{aligned} \tag{3.10}$$

Let $\mu_k = {\alpha_k}^{-1}$, the relative step length for AGD iteration is constructed as follows,

$$\mu_k = \Big(\frac{R_k}{\|p_k\|}\Big)^{-1} = \frac{\|g_k\|}{R_k}. \tag{3.11}$$

In consequence, combining the search direction from (3.10) and the relative step length from (3.11), AGD iteration is created as follows,

$$W_{k+1} = W_k - [\mu_k I + H_k]^{-1} g_k. \tag{3.12}$$

The relative step length μ_k helps to solve indefinite Hessian problem with the following effects: (a) for large μ_k, it approximates the linear optimization of approximate gradient descent to propagate faster towards the minimum point; (b) for small μ_k, it approximates the quadratic optimization of Newton method for faster convergence. Theoretically, the relative step-length μ will exhibit the monotonic decrease behaviour with norm gradient approaching zero, $\|g_k\| \to 0$ progressively due to the nature of iterative equation. As a result, AGD will automatically switch from (a) phase-I to (b) phase-II with enough training iterations and when the training is closed to the minimum point. Therefore, AGD is implicitly a two-phase approach with the addition of relative step-length μ in the update equation. Moreover, the relative step-length is derived from optimal condition of equation and thus providing extra bandwidth to adjust the radius of the spherical search regions for a more appropriate and logical values. The use of relative step-length also enables the construction of more adaptive iterations and is proven to be more robust than the ad-hoc approach used in the Levenberg-Marquardt method [63].

3.3 Stochastic Diagonal Approximate Greatest Descent

In the artificial neural network weight update scheme, the original AGD has to be modified into stochastic manner to introduce noisy gradient. The stochasticity of

training data can reduce the over-fitting issue and impose data generalization during the training process. The modified version of AGD adopts two Hessian approximations, i.e. (a) drop off-diagonal terms of Hessian with respect to weights [67, 68] and, (b) apply truncated Hessian approximation to ignore higher-order differential terms [39]. As a result, the modified version of AGD, dubbed as Stochastic Diagonal AGD (SDAGD) can be realized in DNN training with faster convergence and promising recognition rate. In original AGD algortihm, the full Hessian matrix [69] is written as follows,

$$H = \frac{\partial^2 E}{\partial w_{ij} \partial w_{rs}}, \tag{3.13}$$

where w_{ij} and w_{rs} are the connection weight from unit i to unit j and unit r to unit s, respectively. Since there are no intra-connection between the neurons at the same layer [44], the off-diagonal terms of the Hessian matrix is dropped and lead to,

$$H = \frac{\partial^2 E}{\partial w_{ij}^2}. \tag{3.14}$$

The local second-order derivative of $\frac{\partial^2 E}{\partial w_{ij}^2}$ from (3.14) can be computed with product rule and chain rule from (2.3) to yield,

$$\begin{aligned} \frac{\partial^2 E}{\partial w_{ij}^2} &= \frac{\partial}{\partial w_{ij}} \Big(\frac{\partial E}{\partial w_{ij}} \Big) \\ &= \frac{\partial}{\partial w_{ij}} \Big(\frac{\partial E}{\partial Y} \frac{\partial Y}{\partial w_{ij}} \Big) \\ &= \frac{\partial Y}{\partial w_{ij}}^T \frac{\partial Y}{\partial w_{ij}} + \frac{\partial E}{\partial Y}^T \frac{\partial^2 Y}{\partial {w_{ij}}^2}. \end{aligned} \tag{3.15}$$

However, the diagonal Hessian matrix still suffers from unexpected negative terms that might affect the search direction. Therefore, truncated Hessian from Gauss-Newton is used to guarantee that the approximation is non-negative by clipping off the higher order differential terms. This is equivalent to assuming that the network is a linear function of the parameter. Hence, truncated Hessian approximation is approximated to be the square of the Jacobian matrix which is a positive semi-definite matrix written as,

$$\begin{aligned} \tilde{H} &\approx \frac{\partial Y}{\partial w_{ij}}^T \frac{\partial Y}{\partial w_{ij}} \\ &\approx {G_k}^T G_k. \end{aligned} \tag{3.16}$$

Substituting μ_k with relative step length, the weight update rule in SDAGD algorithm is written as,

$$W_{k+1} = W_k - [\mu_k I + \tilde{H}_k]^{-1} g_k. \tag{3.17}$$

3.4 Convergence Analysis of Stochastic Diagonal Approximate Greatest Descent

Lyapunov function theorem [70] is an important tool used in the analysis of the stability of non-linear dynamical systems described by systems of differential equations, difference equations and functional equations. The following *lemma* presents the criteria of Lyapunov stability theorem concerning the stability of a solution near to a point of equilibrium.

Lemma 3.4.1 *Consider the following iterative equation*

$$w_{k+1} = h(w_k), k = 0, 1, 2, ...,$$

where $h(w_k)$ is a vector of continuous function which is not explicitly dependent on the time variable k and $w^* = h(w^*)$. Then w^* is globally convergent if

1. *There exists a continuous non-negative Lyapunov function $V(w) > 0$ for $w_k \neq w^*$ and $V(w^*) = 0$;*
2. *All level sets of $V(w)$ are properly nested (topologically equivalent to a concentric spherical surfaces); and*
3. *$\Delta V(w_k) = V[h(w_k)] - V(w_k) < 0$ for $w_k \neq w^*$ and $\Delta V(w^*) = 0$.*

Subsequently, the following *lemma* is used to prove the convergence of numerical method in artificial neural network optimization. The relationship between numerical optimization update rule and the conditions for Lyapunov function theorem are defined.

Lemma 3.4.2 *Suppose that*

1. *The function $f(w) \in C^2$ has a unique minimum point w^* in the region $\Omega = \{w|f(w) \leq K\}$, where K is an arbitrary large positive number;*
2. *The level sets of the function $f(w)$ are properly nested globally; and*
3. *The Lyapunov function $\Delta V(w) = f(w) - f(w^*)$.*

In solving optimization problem, let the objective function as the choice of Lyapunov function $V(w)$ where

$$V(w) = f(w) - f(w^*). \tag{3.18}$$

From the expansion of (3.18) yields,

$$V(w_k) = f(w_k) - f(w^*), \tag{3.19}$$

and

$$V(w_{k+1}) = f(w_{k+1}) - f(w^*). \tag{3.20}$$

The difference of equations in (3.19) and (3.20) produces,

$$\begin{aligned} V(w_{k+1}) - V(w_k) &= f(w_{k+1}) - f(w_k), \\ \Delta V(w_k) &= \Delta f(w_k). \end{aligned} \tag{3.21}$$

Since $V(w)$ is a Lyapunov function as shown in *lemma* 3.4.2, the condition $\Delta V(w_k) = \Delta f(w_k) < 0$ for all $w_k \neq w^*$, where $w \in \Omega$ and $\Delta V(w^*) = 0$. Subsequently, the convergence of SDAGD algorithm is analyzed using the Lyapunov function theorem as shown in the *theorem* 3.4.3 .

Theorem 3.4.3 *The objective function $f(w)$ of ANN is continuous differentiable. Ultimately, the optimization of SDAGD algorithm should achieve,*

$$\Delta f(w_k) = f(w_{k+1}) - f(w_k) < 0.$$

Proof From the Taylor series expansion of $f(w_k)$,

$$f(w_{k+1}) \approx f(w_k) + g(w_k)^T(w_{k+1} - w_k) + O({w_k}^2), \tag{3.22}$$

where $O({w_k}^2)$ is the higher order element. The higher order expansion is truncated here for gradient approximation, hence,

$$f(w_{k+1}) \approx f(w_k) + g(w_k)^T(w_{k+1} - w_k). \tag{3.23}$$

Rearrange,

$$\begin{aligned} f(w_{k+1}) - f(w_k) &= g_k^T(w_{k+1} - w_k), \\ \Delta f(w_k) &= g_k^T \Delta w_k. \end{aligned} \tag{3.24}$$

Since, $w_{k+1} = w_k - \left[\mu_k + g(w_k)^T g(w_k)\right]^{-1} g(w_k)$. Hence,

$$\Delta w_k = -\left[\mu_k + g_k^T g_k\right]^{-1} g_k. \tag{3.25}$$

Therefore, let $\tilde{H}_k = {g_k}^T g_k$ be the diagonal square of the Jacobian, $\tilde{H}$ is always positive and substitute (3.25) into (3.24) will achieve,

$$\begin{aligned} \Delta f(w_k) &= -g_k^T g_k \left[\mu_k + g_k^T g_k\right]^{-1} \\ &= -\tilde{H}_k \left[\mu_k + \tilde{H}_k\right]^{-1} \\ &= -\left[\mu_k \tilde{H}_k^{-1} + 1\right]^{-1}. \end{aligned} \tag{3.26}$$

Since $\mu_k = \frac{\|g_k\|}{R_k}$ utilizes user defined radius hyperparameter with $R_k > 0$ and $\tilde{H}_k$ is the diagonal version of squared Jacobian, the SDAGD's component is always positive definite. Hence,

$$\Delta f(w_k) = -\left[\mu_k \tilde{H}_k^{-1} + 1\right]^{-1} < 0. \tag{3.27}$$

As stipulated in *lemma* 3.4.2, the convergence of SDAGD algorithm is guaranteed if

$$\Delta V(w_k) = \Delta f(w_k) < 0 \quad \forall \quad w \in \Omega. \tag{3.28}$$

As a conclusion, the convergence of SDAGD algorithm fulfills the condition of Lyapunov stability theory, and hence it is proven that the proposed algorithm ensures the weight convergence and stability at all time.

3.5 Experiment #1: SDAGD with Shallow Neural Networks

To test on the feasibility of SDAGD algorithm in artificial neural networks training, a two-layer Multilayer Perceptron (MLP) as shown in Figure 3.3 with one hidden layer and one output layer is designed to classify MNIST dataset. MNIST [44] database as depicted in Figure 3.4 comprises of handwritten digits from 0 to 9 with $60,000$ training images and $10,000$ testing images in a dimension of 28×28 bounding boxes. MNIST dataset is suitable for prototyping new algorithms or network architectures due to less complexity of the dataset. Each pixel of MNIST's gray-scale input image is rearranged as vector form and feed into the input of the network through incremental mode. Subsequently, the network provides the inference with the highest probability out of the multiple classes of outputs. The network structure is designed with 784 input nodes, 700 hidden nodes and 10 output classes. The sigmoid activation function is used for non-linear activation and the network is trained with SGD [19], Stochastic Diagonal LM (SDLM) [44] and SDAGD optimizers for performance comparison.

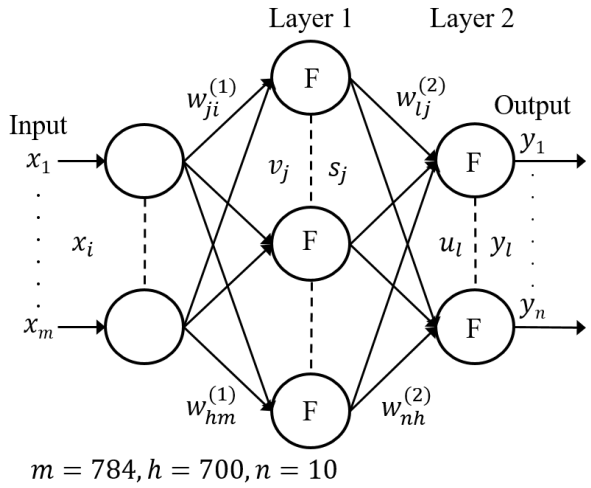

Figure 3.3: General architecture of two-layer multilayer perceptron with a single hidden layer and an output layer.

Figure 3.4: Examples of MNIST images of handwritten digits from 0 to 9.

In consequence, two performance measures are selected for the experiment evaluation, i.e. misclassification rate (MCR) and Mean Squared Error (MSE). The objective function tested in all the experiment is the MSE, calculated as follows,

$$MSE = \frac{1}{P}\sum_{a=1}^{P}\sum_{b=1}^{n}\left(d_{b,a} - x_{b,a}\right)^2, \tag{3.29}$$

where $d_{b,a}$ is the targeted output, $x_{b,a}$ is the output of the network and P represents the total number of inputs. Subsequently, MCR is formulated as follows,

$$MCR = \left(1 - \frac{\sum TP + \sum TN}{\sum TS}\right) \times 100\%, \tag{3.30}$$

where TP represents true positive, TN represents true negative and TS is the total number of samples. The termination criteria of the training process is determined when

one of the two criteria is fulfilled, i.e. the max epoch is achieved or when the MSE value is less than 0.01.

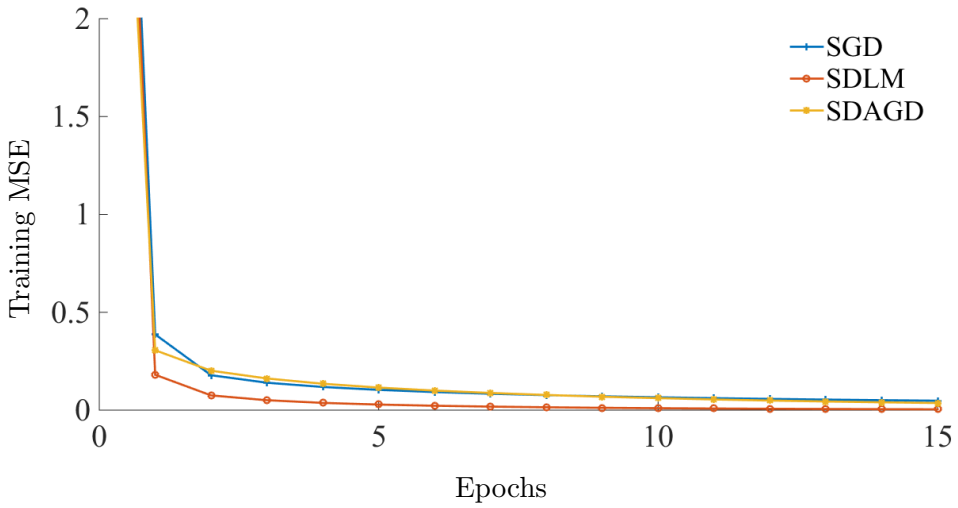

Figure 3.5: Training mean squared error of shallow neural networks.

Figure 3.5 demonstrates the training curves of SGD, SDLM and SDAGD optimizers on a two-layer MLP. SGD algorithm performs worst in terms of MSE when compared to other Hessian-based variants (SDLM and SDAGD). In contrast, Hessian-based approach with local curvature information are able to demonstrate steeper training curve in comparison to gradient-based approach. This observation suggests that gradient-based approach may not work as efficiently as Hessian-based approach without the geometrical information of the error surface. Apart from that, the training curve of SDLM is steeper than SDAGD algorithm. However, the final MCR of SDLM (4%) is lower than SDAGD algorithm (3.34%) indicates that SDLM has caused over-fitting problem.

Table 3.1: Testing misclassification rate of different learning algorithms.

Training Algorithm	Configuration	MCR(%)	σ (%)
SGD	$\eta = 0.01$	6.68	0.1001
SGD	$\eta = 0.1$	3.81	0.1773
SDLM	$\eta = 0.001$, $\mu = 0.01$	4.00	0.1147
SDAGD	$R = 0.1$	3.34	0.142

Table 3.1 tabulates the result of testing MCR with regards to SGD, SDLM and the

SDAGD optimizers with the standard deviation, σ of less than 0.2% out of the three runs. For SDAGD algorithm, the network yielded a lower testing MCR when compared to SGD and SDLM algorithms. This result also demonstrates how AGD's long-term optimal trajectory control is improving the recognition rate by using two-phase switching strategy. Another sub-experiment is conducted to further verify the choice of learning rate on SGD algorithm with respect to testing MCR. Table 3.1 demonstrates that a properly tuned SGD algorithm with learning rate at 0.1 that could provide the better testing MCR than SDLM algorithm. However, SDAGD achieves lower MCR with the long-term optimal trajectory with the help of relative step length. The use of relative step length in AGD provides better roll-off rate by adaptively switching from phase-I to phase-II depending on iterations trajectory to the optimal solution. In terms of average execution time per epoch, SDAGD algorithm is taking longer time to compute the Hessian information in each iteration. However, the computation inefficiency is compensated with steeper roll-off and higher recognition rate acquired in fewer iterations. As a result, SDAGD algorithm with lower MCR rate and steeper training curve than other existing optimizer increases the performance of the shallow neural network. Hence, the implementation of second-order derivative and curvature information through an adaptive manner is able to avoid over-fitting problem and the needs of hyperparameter fine-tuning.

3.6 Experiment #2: Weight Convergence of SDAGD

The changes of weights responded to training algorithms is investigated numerically to observe the weight convergence. Two-layer neural network is set up to record the change of weight parameters. The sum of weights is observed along the training epoch with the equation as follows,

$$\sum W_{epoch} = \frac{1}{N} \sum_{i}^{a} \sum_{j}^{b} (|w_{ij,epoch}^{(l)}|), \tag{3.31}$$

where $N = a^{(l)} \times b^{(l)} + a^{(l+1)} \times b^{(l+1)} ... a^{(L)} \times b^{(L)}$ represents the total number of weights, a is the total number of weights (hidden and output), b is the total input nodes at the respective layer and let $l = 1, 2, .., L$.

Figure 3.6 demonstrates the summation of all weights in the network with one hidden layer using SDAGD, SGD and SDLM. As a result, the network weights using SDAGD algorithm responded with the steepest change at the first few epochs, while the other

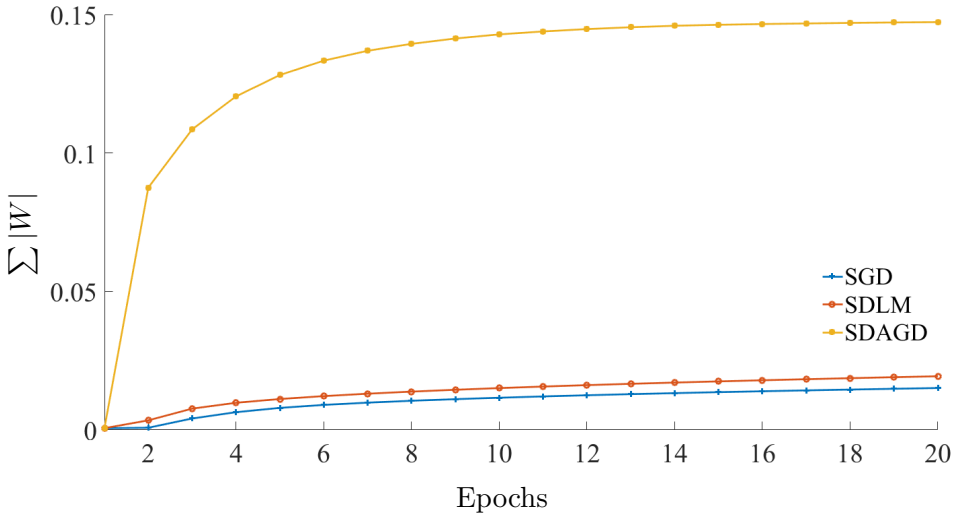

Figure 3.6: Sum of weight parameters with respect to epoch to verify the weight convergence of SGD, SDLM and SDAGD.

methods exhibit small changes. The result indicates that the training algorithms converged to two distinctive optimal weights, which is unlikely to be observed from high dimensional objective function. In this experiment, the effect of long-term optimization strategy provided by SDAGD algorithm is demonstrated with rapid descent at the beginning of the training phase with the large step-length. Subsequently, SDAGD algorithm enables the switch from AGD to approximate Newton method when the step-length is approaching to zero. As compared to SDAGD algorithm, SGD and SDLM methods converged at another point of optimal weight with less drastic weight changes in the network. Therefore, SDAGD algorithm has obtained the lowest misclassificaiton rate in testing phase due to the long-term optimization strategy for optimal weights searching in the training phase.

3.7 Experiment #3: Radial Step-length Effect of SDAGD

To evaluate the sensitivity of radius hyperparameter, R of the spherical search regions in SDAGD algorithm, a similar two-layer MLP with 784 input nodes, 700 hidden nodes and 10 output nodes with sigmoid activation function is selected for performance evaluation. The network is trained with MNIST dataset up to 15 epochs. The training dataset is segregated into mini-batches with the size of 100 images per batch. This experiment focused mainly on the following aspects, (a) the radial effect of different

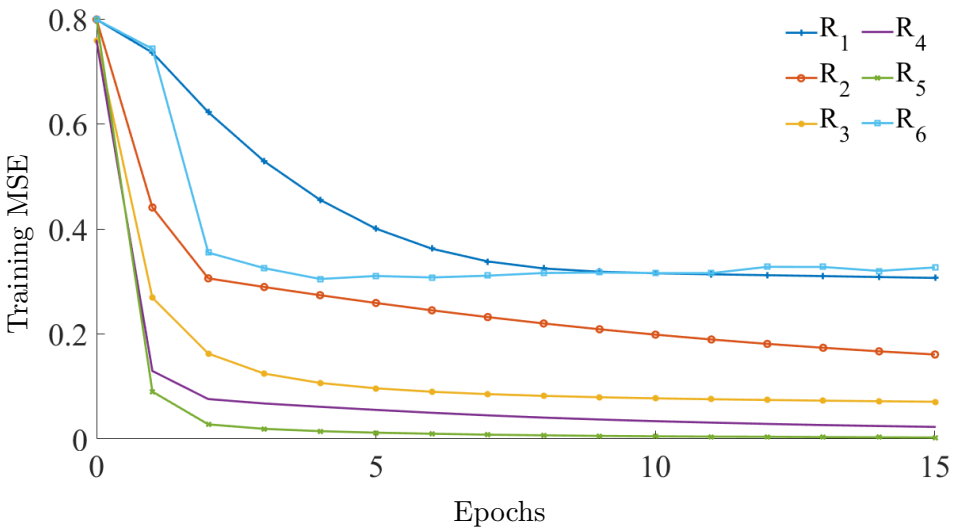

Figure 3.7: Training mean squared error at varying R with respect to epoch.

spherical search regions in response to MSE and MCR, (b) the effects of annealing radius hyperparameter. To test the radial effect of SDAGD, the following values of radius hyperparameter is tested, i.e. $R_1 = 0.0001$; $R_2 = 0.001$; $R_3 = 0.01$; $R_4 = 0.1$; $R_5 = 1$ and $R_6 = 10$.

Figure 3.7 depicts the training curve responded to each radius settings of R against training error rate reflected from training set. All training curves demonstrated gradual descent of MSE resembling the behavior of most optimizers. However, only settings with R_3, R_4 and R_5 possess steeper training curve towards the solution and remained constants across the rest of the epochs. This response indicates a decent size of local spherical search regions is used for iteration construction. As for R_1 and R_2, the training curves simulated overly small spherical search regions and failed to converge fast enough to a region nearer to the optimal solution. In consequence, more epochs are needed to achieve lower MSE. R_6 on the other hand, experienced increase of MSE at the fifth iteration onward. The response demonstrates overly large spherical search regions and therefore diverging towards other bad local minima. As a result, the radial step-length settings of radius hyperparameter should be bounded between R_3 and R_5 for optimal performance.

Figure 3.8 plots the radial effect of radius parameters in response to changes in testing misclassification rate. This graph exhibits similar behavior to training error. The ideal settings of radius hyparameter lie between R_3 to R_5. Other values of R

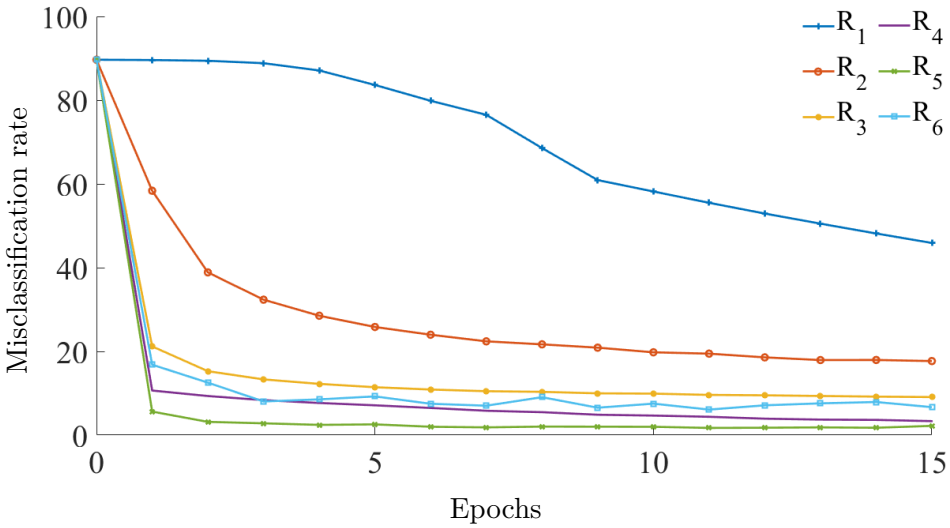

Figure 3.8: Testing misclassification rate at varying R with respect to epoch.

indicates either too large or too small of the choice of local spherical search regions during training phase, which in turn results in worst MCRs. Besides, having the MCR results closely resembling the training curve suggests that the network is not over-fitted. Moreover, from the bathtub-like response recorded based on the final MCR of each setting in Figure 3.9, the radius hyperparameter should be set between R_3 to R_5 for optimal performance. As a result, SDAGD algorithm is able to work with large range of hyperparameter setting and are more robust to fine-tuning hyperparameter activities.

Annealing radius hyperparameter is proposed to increase the sensitivity of relative step length. By annealing the radius of the spherical search regions, SDAGD performs normalization to scale up the relative step-length to explore optimal solution at the optimization level set of phase-II. The value of radius hyperparameter (R) is gradually reduced across training epochs. In this experiment, annealing radius hyperparameter is tested with the similar two-layer MLP to observe the change of MSE and MCR. Annealing radius hyperparameter is performed at every epoch via the following formula,

$$R_{anneal} = R_{min} + (R_{max} - R_{min}) \times {R_{fade}}^{epoch}, \tag{3.32}$$

where $R_{min} = R_4$, $R_{max} = R_5$, $R_{fade} = 0.8$ and *epoch* resembles the number of iterations. The determination of value R_{min}, R_{max} and R_{fade} is defined from the Figure 3.8.

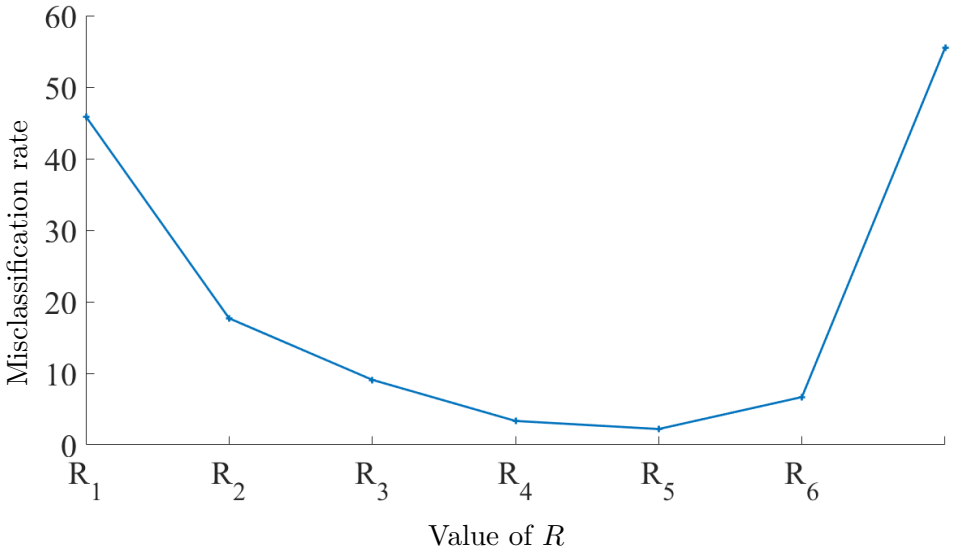

Figure 3.9: The misclassification rate with respect to R for annealing radius effect.

Table 3.2: Training error and testing misclassification rate for fixed R and R_{anneal}.

Configuration	MSE	MCR(%)
R_5	0.0028	2.21
R_{anneal}	0.0023	1.62

Table 3.2 tabulated the MSE and MCR comparing both annealing radius hyperparameter and the best result from previous experiment (R_5). Annealing radius hyperparameter achieved better MSE and MCR than static value of R. The result shows the benefits of monotonic decrease in step length resulted in better training MSE and testing MCR. This is due to larger iteration steps at the beginning of learning curve for faster roll-off and gradually reduced to more suitable radius for rapid convergence. The radial step-length effect in SDAGD algorithm has a role to determine appropriate step-length that possesses great influences to the overall network result. The experiment concluded that the value of R is directly proportional to the learning rate of SDAGD algorithm when R is bounded between R_3 and R_5.

3.8 Chapter Summary

Adaptive learning rate approach using Hessian-based Approximate Greatest Descent (AGD) method is proposed to optimize the weight of shallow neural networks. AGD use the concept of optimal control to derive long-term optimal trajectory for weight parameter optimization. Subsequently, Stochastic Diagonal Approximate Greatest Descent (SDAGD) is proposed to reduce the computation cost and memory dependency in the backpropagation manner using two Hessian approximations: (a) applying truncated Hessian and (b) omitting the off-diagonal terms. In phase-I, SDAGD performs multiple spherical search regions construction to determine the relative step-length to seek the minimum points at the boundary. When the relative step-length is closer to zero, it switch from AGD to Newton method to optimize the weight parameter in phase-II. The convergence analysis of the proposed algorithm is verified using the Lyapunov function theorem. Network weight summation is demonstrated numerically to show the convergence of error. As a result, SDAGD converged to a better optimal solution as compared to SGD and SDLM with lower testing MCR, which is 3.34%. Throughout the experiment with shallow neural network, SDAGD algorithm demonstrated better training trajectory and performance compared to the SGD and SDLM. From the radial step-length effect experiment, SDAGD algorithm is tested with annealing radius setting between 0.001 to 1 and it achieved lower MCR of 1.62% compared to fixed radius setting of 2.21%. Furthermore, incorporating annealing radius hyperparameter can scale up the relative step-length iteratively from the initial search region to the final optimum solution level set. The investigation of SDAGD in shallow neural networks provides the evidence of good performance. Hence, the development of SDAGD in deep learning neural networks will be tested in the next chapter to evaluate the performance and the issue of vanishing gradient mitigation.

Chapter 4

Vanishing Gradient Mitigation Analysis in Deep Learning Neural Networks

4.1 Introduction

Deep learning neural network originates from multi-layered feedforward artificial neural network, or more formally known as the Multilayer Perceptron (MLP). Unlike MLP, DNN can perform various functions, e.g., synthesis/generation or recognition/classification depending on the architectures and learning techniques. Generally, there are three types of deep learning architectures [2], namely: (a) generative deep architecture, (b) discriminative deep architecture and (c) hybrid deep architecture. Generative deep architecture uses unsupervised learning to capture high-order correlation of the observed data for pattern analysis or synthesis purposes. These architectures include Stacked Auto-Encoder [71], De-noising Auto-Encoders [72], Deep Boltzmann Machine [73], Deep Belief Network [4], Sum Product Network [74, 75] and Recurrent Neural Network [18]. Alternatively, a discriminative deep architecture uses supervised learning manner to discriminate visible data with the targeted labelled data, for e.g., Deep Structured Conditional Random Field [7, 76], Convolution Neural Network [77, 78] and Deep Stacking Network [79–81]. However, artificial neural networks often experience training problem due to vanishing and exploding gradient [1, 28, 82]. The training problem is amplified exponentially especially in deep learning. Deep learning refers to a

complex artificial neural networks architecture, where many layers of information processing stages in hierarchical architectures are trained via backpropagation algorithm for pattern classification [1].

Backpropagation algorithm utilizes optimization techniques to compute the error function. Then, the loss function is backpropagated from output layer to the input layer for weights parameter fine-tuning. Standard backpropagation fails to perform in deep neural networks due to the pervasive presence of local optima and other optimization challenges in the non-convex objective function [28]. This problem has driven the top hidden layers into saturation and is becoming more severe as the depth increases. The main culprit of vanishing or exploding gradient is the choice of activation function. Activation function can be further divided into saturated activation function and unsaturated activation function. Saturated activation functions are often used for decision boundaries in the early years of neural network developments. Sigmoid function [83] is popular because of its differentiable property that is suitable for backpropagation in neural networks. In contrast, rectified linear unit (ReLU) [84] is an example of unsaturated activation functions known to overcome saturation problem. ReLU avoids the usage of exponential term for non-linearity, therefore free from vanishing gradient problem. ReLU is currently the most commonly used activation function for artificial neural network training. Nevertheless, SDAGD emerges as an alternative to the currently available adaptive optimization techniques. SDAGD algorithm is proposed to adopt the concept of long-term optimal trajectory from dynamical control theories into deep learning optimization. SDAGD algorithm utilizes the concept of relative step-length to modify the original step-length alongside the training iteration. In this chapter, the utilization of SDAGD algorithm is applied to deep neural network with different activation functions to solve the vanishing gradient problem.

This chapter covers an overview of deep learning neural networks with brief introduction to activation functions. Section 4.2 breakdowns the activation function into two categories, i.e. saturated and unsaturated activation functions. Subsequently, the behavior of the proposed method in three common error topographies are covered in Section 4.3. Section 4.4 presents the rationale of vanishing gradient mitigation with the proposed method. Lastly, the chapter summary is covered in Section 4.5.

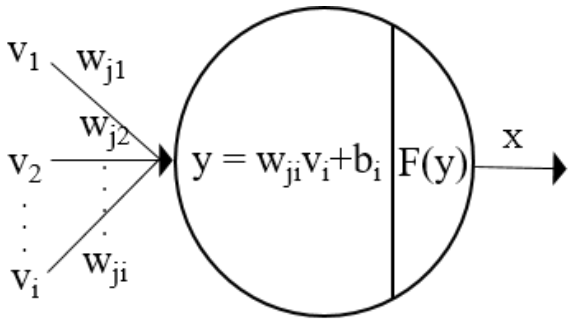

Figure 4.1: Basic operation of a perceptron in artificial neural network.

4.2 Activation Functions

Deep learning neural networks consist of multiple hidden layers stacked up in a hierarchical manner to compute inference. Each layer is made up of multiple artificial perceptrons or nodes as shown in Figure 4.1. Upon summing up all the product of input and weight parameters, an activation function is applied to determine the firing of each neuron with the equations shown below,

$$y = \sum_{i=1}^{n} w_i \times v_i + b, \tag{4.1}$$

$$x = F(y), \tag{4.2}$$

where n represents the total number of nodes, w_i refers to the weight parameters, v_i is the inputs, b is the bias parameters and $F(\cdot)$ is the activation function. The activation function provides non-linearity to the sets of input and limits the boundary of the firing to a finite value [85]. The fired neurons are activated in each layer, therefore the network is able to provide the correct inference after a series of training iterations. There are two types of activation function in general, i.e. saturated and unsaturated activation functions.

4.2.1 Saturated Activation Function

Logistic function is the classical activation function used in the field of neural networks due to close resembling of biological activation rate [83]. There are two types of logistic function, i.e. sigmoid and hyperbolic tangent (tanh) function. Sigmoid activation function is more commonly used in early neural networks era. The logistic

activation function exhibits a 'S' shape response with the balance between linear and non-linear responses. Sigmoid activation is computed via,

$$F_1(x) = \frac{1}{1 + e^{-x}}. \tag{4.3}$$

Moreover, the hyperbolic tangent activation function can be written as,

$$F_2(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{4.4}$$

In consequence, the firing of each neuron is bounded between 0 and 1 for sigmoid activation function and -1 to 1 for tanh activation function. The switching between the two upper and lower boundaries are theoretically identical to switching on/off a neuron. However, the gradient of the data fallen close to either the lowest or highest boundaries are almost zero. Passing saturated zero gradients through backpropagation into the networks causes loss of information. Hence, the gradient of each layer vanishes across iterations with saturated activation function. In consequence, the training iteration makes little to zero progress due to the loss of error signal for optimization. There are other variance of logistic function such as multistate activation function (MSAF) [86]. MSAF combines logistic functions with a constant r_1 to create additional output states. There are two versions of MSAF, i.e. N-order MSAF and the symmetrical MSAF. A 2-order MSAF can be written as,

$$F_3(x) = \frac{1}{1 + e^{(-x - r_1)}} + \frac{1}{1 + e^{-x}}, \tag{4.5}$$

which has an output fluctuate between the three states, i.e. 0 , 1 and 2. Moreover, the symmetrical MSAF is written as,

$$F_4(x) = -1 + \frac{1}{1 + e^{(-x - r_1)}} + \frac{1}{1 + e^{-x}}, \tag{4.6}$$

that also has an output of three states i.e. -1, 0 and 1. Despite the differences, the additional states in MSAF help to dilute the saturation towards the lower or upper boundaries to prevent vanishing gradient problem.

4.2.2 Unsaturated Activation Function

Rectified linear unit (ReLU) [84] is an example of unsaturated activation function introduced to overcome the vanishing gradient problem. Since most state-of-the-art

network architectures are often deep and wide, ReLU is the most commonly used activation function nowadays. ReLU returns 0 if it receives negative values while returning the input values for any positive values, or more conveniently written as,

$$F_5(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0. \end{cases} \tag{4.7}$$

ReLU is also known as the ramp function since the output is either 0 or some positive values. As a consequence, ReLU is prone to exploding gradient problems since the output is not regulated for any positive values [87]. Hence, the implementation of ReLU activation function is usually incorporated with proper weight parameters initialization and/or the pre-training algorithms. Moreover, providing zero as output during negative input will have a zero gradient passing through the backpropagation algorithm. This causes the neuron to die (prevent further firing from the particular neurons) and will never be reactivated again. To avoid dying ReLU problem, Leaky ReLU (LReLU) [88] proposed a non-zero slope value, b to the negative part of ReLU and thus avoiding backpropagating zero gradient into the networks. LReLU can be written as,

$$F_6(x) = \begin{cases} x, & x > 0 \\ bx, & x \leq 0. \end{cases} \tag{4.8}$$

However, LReLU is reported to have inconsistent results depending on the choice of the non-zero slope value which then solved by Parametric ReLU (PReLU) [8] . PReLU assigns the slope value as a training parameter rather than specifying a predefined constant value as in LReLU.

4.2.3 Summary of Saturated and Unsaturated Activation Function

Although unsaturated activation function is less prone to vanishing gradient problem, exploding gradient and dead neuron issues are still discouraging. Network setup with unsaturated activation function requires more attention in initial weights initialization, thus adding more network hyperparameters for fine-tuning will further increases the complexity of the neural network. Nevertheless, saturated activation function is still popular due to the smoother non-linearity squashing between upper and lower boundary. The output vastly resembles probabilities thus showing clearer traces of activated neurons and is particularly useful during troubleshooting. On the other hand, saturated activation causes vanishing gradient when the derivative of higher hierarchy layers approaches zero. As a result, there is increased popularity in research towards the effects of adaptive optimization techniques with vanishing gradient issues.

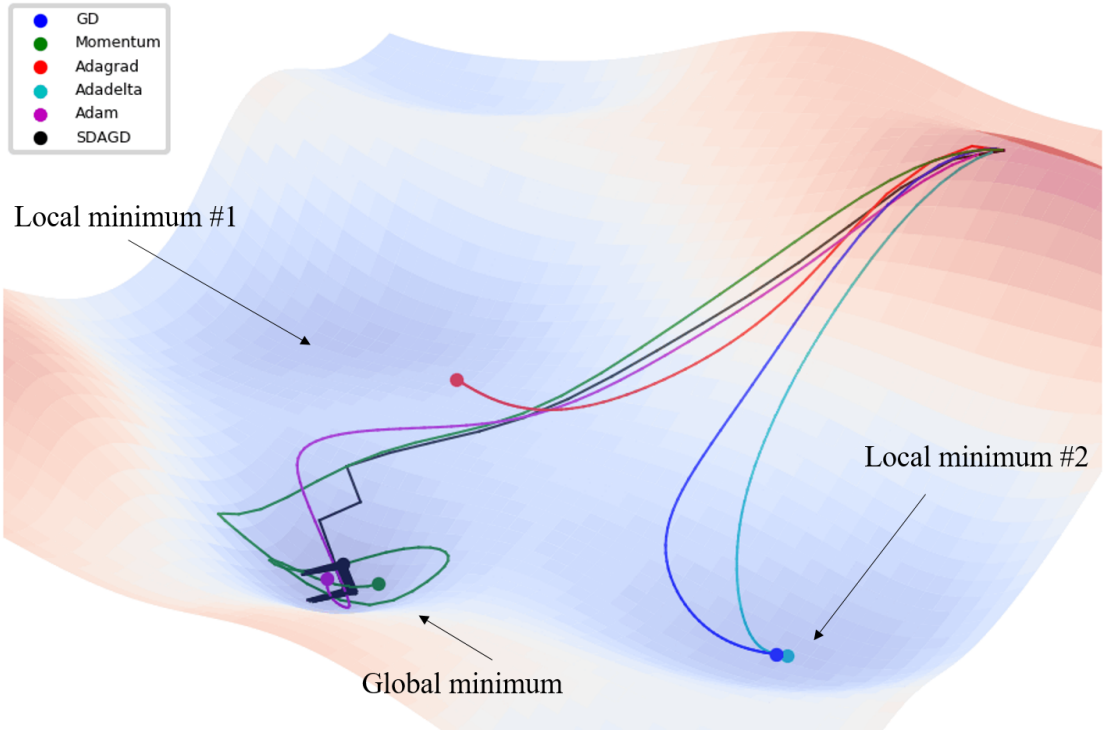

Figure 4.2: Hilly error surface with two local minima and one global minimum.

4.3 Experiment #1: Optimization Visualization

In order to visualize the working mechanism of each optimizer, three optimization problems are setup by utilizing bivariate normal distribution [89]. The optimization problems aimed to simulate how each optimizer is adopting different strategies in seeking for an optimal solution: (a) a hilly error surface with two local minima and one global minimum; (b) a deep Gaussian trench to simulate drastic gradient changes experienced with ravine topography and (c) small initial gradient to simulate a plateau terrain. The hilly topography for problem (a) and (b) is constructed using (4.9) as follows,

$$T(x, y) = -\sin x^2 \cos 3y^2 e^{-(xy)^2} - e^{-(x+y)^2}. \tag{4.9}$$

In addition, the Gaussian trench is superimposed with $T(x, y)$ as follows,

$$G(x, y) = -e^{\frac{(x-a)^2}{2c^2} + \frac{(y-b)^2}{2c^2}}, \tag{4.10}$$

where x and y are the input, a and b determine the location of the trench on the error surface and c sets the width of the Gaussian trench. In problem (a), the value c is set as 0.35 while in problem (b) and (c), the value c is set to 0.2. For comparison, each setup

is tested with gradient descent (GD), momentum GD, AdaGrad, AdaDelta, ADAM and the proposed algorithm.

Figure 4.2 depicts the behavior of how each optimizer responded differently even though given the same starting gradient. AdaGrad and AdaDelta descent faster than GD algorithm due to cumulative gradients and per-parameter adaptation, respectively. However, GD, AdaGrad and AdaDelta are having trouble converging to the global minimum and being trapped at the local minima. In contrast, SDAGD, ADAM and Momentum GD makes it to the global minimum. Momentum GD exhibits overshooting behaviour due to accumulated momentum but is able to converge with more iterations. Additionally, the trajectory of how ADAM converges is similar to momentum GD but in a more controlled manner due to the adaptation of second moment estimation. Second moment estimation helps to incorporate local curvature information into the algorithm and thus damping the momentum effects. SDAGD algorithm converges to the global minimum with the least iteration. The long-term optimal trajectory towards the global minimum of SDAGD algorithm is demonstrated with the curvature information derived from the Hessian-based approach. Moreover, the effect of relative step-length in SDAGD

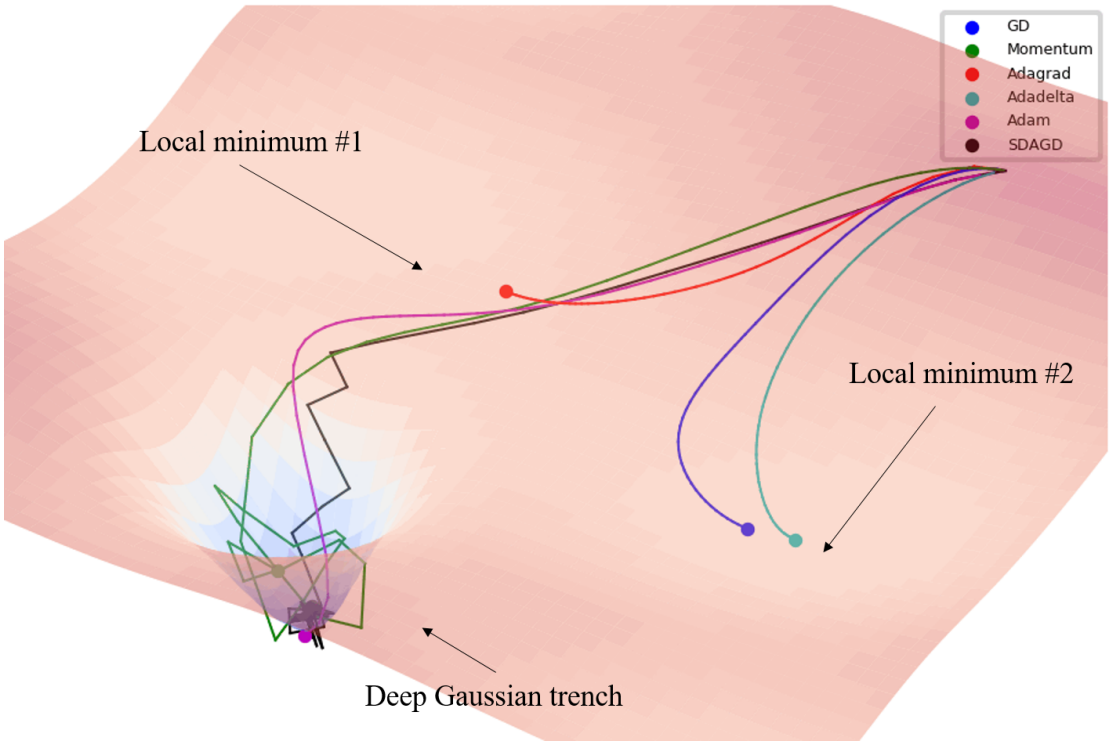

Figure 4.3: Deep Gaussian trench to simulate drastic gradient changes.

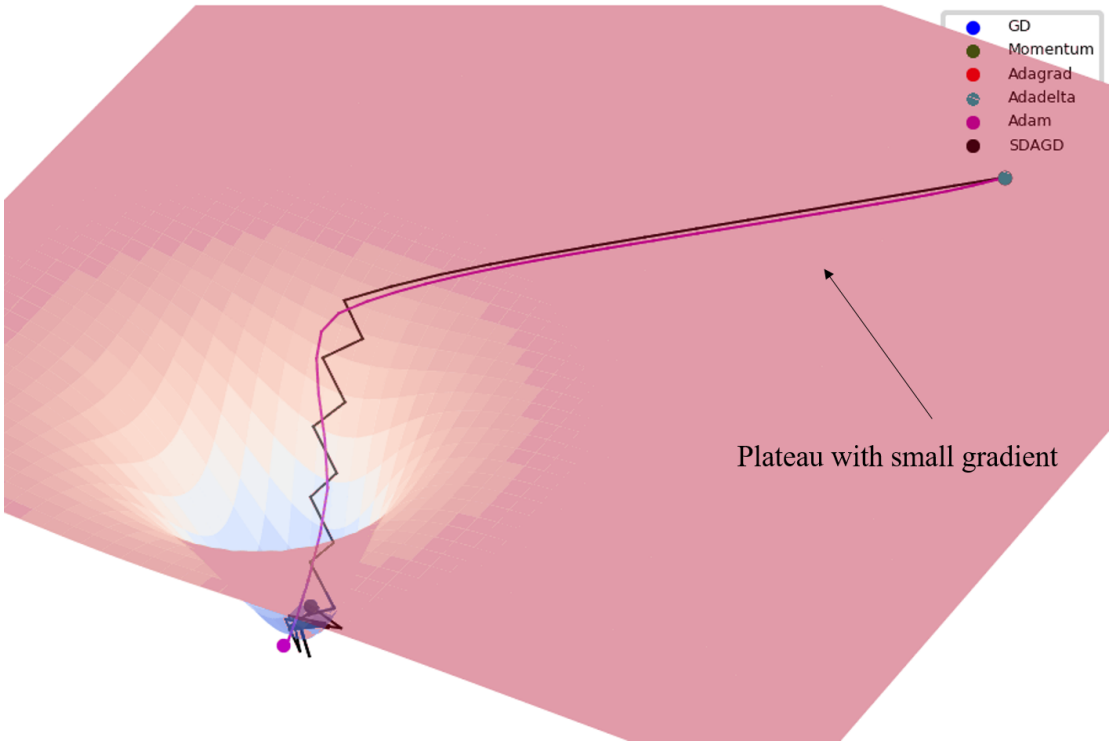

Figure 4.4: Small initial gradient to simulate a plateau terrain.

allows faster convergence by constructing larger spherical search regions initially and reduces monotonically as the iteration approaches the solution.

Figure 4.3 is intended to show how different optimizers react to deep ravines topography. Based on the trajectory of momentum GD, multiple large gradients are added together and eventually causing the trajectory to roll away from the minimum point. This example exhibits a good demonstration of how certain optimizers roll over deep ravines and will never converge to the minimum point within the ravine. In contrast, both SDAGD and ADAM possess controlled trajectory descending due to the local curvature information from the Hessian-based computation. Another common error surfaces is shown in Figure 4.4, it is designed to simulate plateau where the gradients are close to zero. The plateau topography caused other optimizers to stay at the initial point and only ADAM and SDAGD algorithms arrive at the minimum point. The results showcase the importance of the Hessian-based information that promotes propagation despite having minuscule gradients. Nevertheless, SDAGD algorithm still converges to the minimum point in lesser iteration than ADAM in all the experiments vastly due to the effect of long-term optimal trajectory control theory. The relative step-length

in SDAGD algorithm provides a two-phase switching approach with larger steps at the start of training and decays monotonically as the iteration grows.

This experiment covers the behavior of SDAGD algorithm optimizing through three simulated error surfaces. Despite all the challenges, SDAGD algorithm converges to the minimum point faster than other existing optimization algorithms. For instance, the Hessian element in SDAGD algorithm imparts local curvature information into optimization helps prevent trapping in plateau and overrunning ravines. Additionally, the spherical search regions constructed by the relative step length helps to keep the trajectory rolling, thus mitigating the vanishing gradient problem seen in ordinary gradient descent. As a result, the robust trajectory constructed by SDAGD algorithm towards the minimum point across plethora of error surfaces is not confined by the vanishing gradient problem.

4.4 Experiment #2: Comparison of Saturated and Unsaturated Activation Functions with SDAGD

This experiment is conducted to demonstrate the feasibility of applying SDAGD algorithm into deeper neural networks particularly to examine the vanishing gradient issue. Five versions of deep feedforward networks with 784 input nodes, 700 hidden nodes and 10 output nodes are setup to simulate deep learning. Every version of neural network utilizes the same hidden layer configuration, i.e. hl-1, hl-2, hl-3, hl-4 and hl-5 for 1 to 5 hidden layers, respectively. Both sigmoid and ReLU activation functions are used to compare saturated and unsaturated activation functions. For benchmark purposes, MNIST dataset [44] is chosen for reproducibility. The constructed network is then trained with SGD and SDAGD with mini-batch size of 100 inputs per batch for performance comparison. All parameters reported are fine-tuned accordingly to produce the best results possible. Additionally, the reported results are the average value computed out of 3 runs of different sets of initialized weights using Xavier initialization [28].

Figure 4.5 depicts the training curve for SGD and SDAGD algorithms with sigmoid activation. Based on the training curves in Figure 4.5(a), SGD is having difficulties in training starting with hl-2 to hl-3 and fails to train completely in hl-4 to hl-5. SGD with sigmoid activation function is prone to have saturation problem and is used to simulate vanishing gradient for this experiment. Conversely, the proposed SDAGD algorithm

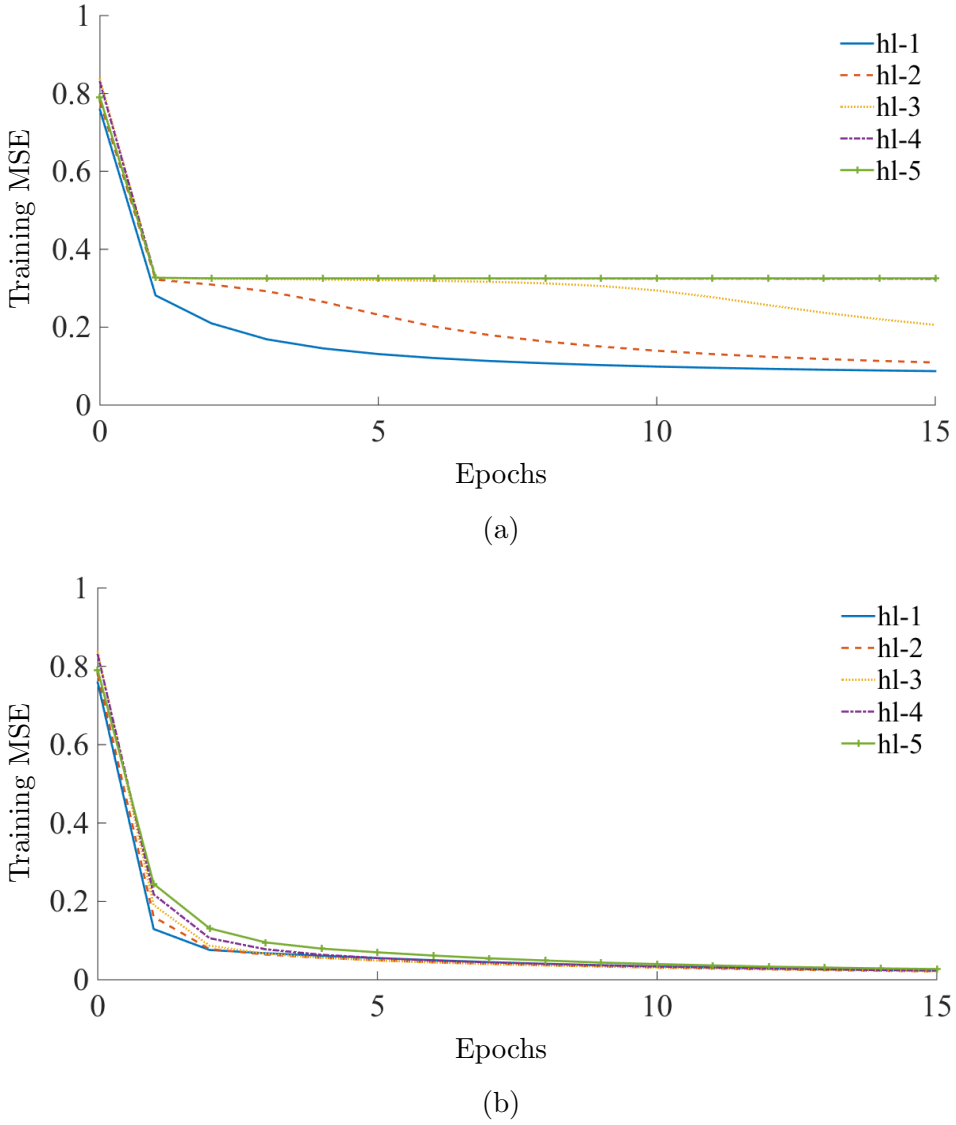

Figure 4.5: Training mean squared error of varying number of hidden layers with sigmoid activation function. (a) Training with SGD and (b) training with SDAGD.

is able to train throughout all models of different hidden layers as shown in Figure 4.5(b). This is due to the two-phase switching approach adopted by SDAGD algorithm to alter the step length adaptively in conjunction with different training phases. Overall, SDAGD algorithm still possesses faster roll-off rate compared to SGD. As for Figure 4.6, it is clear that both SGD and SDAGD algorithms are able to train with ReLU activation function without any issue. However, Figure 4.6(a) shows that SGD struggles from

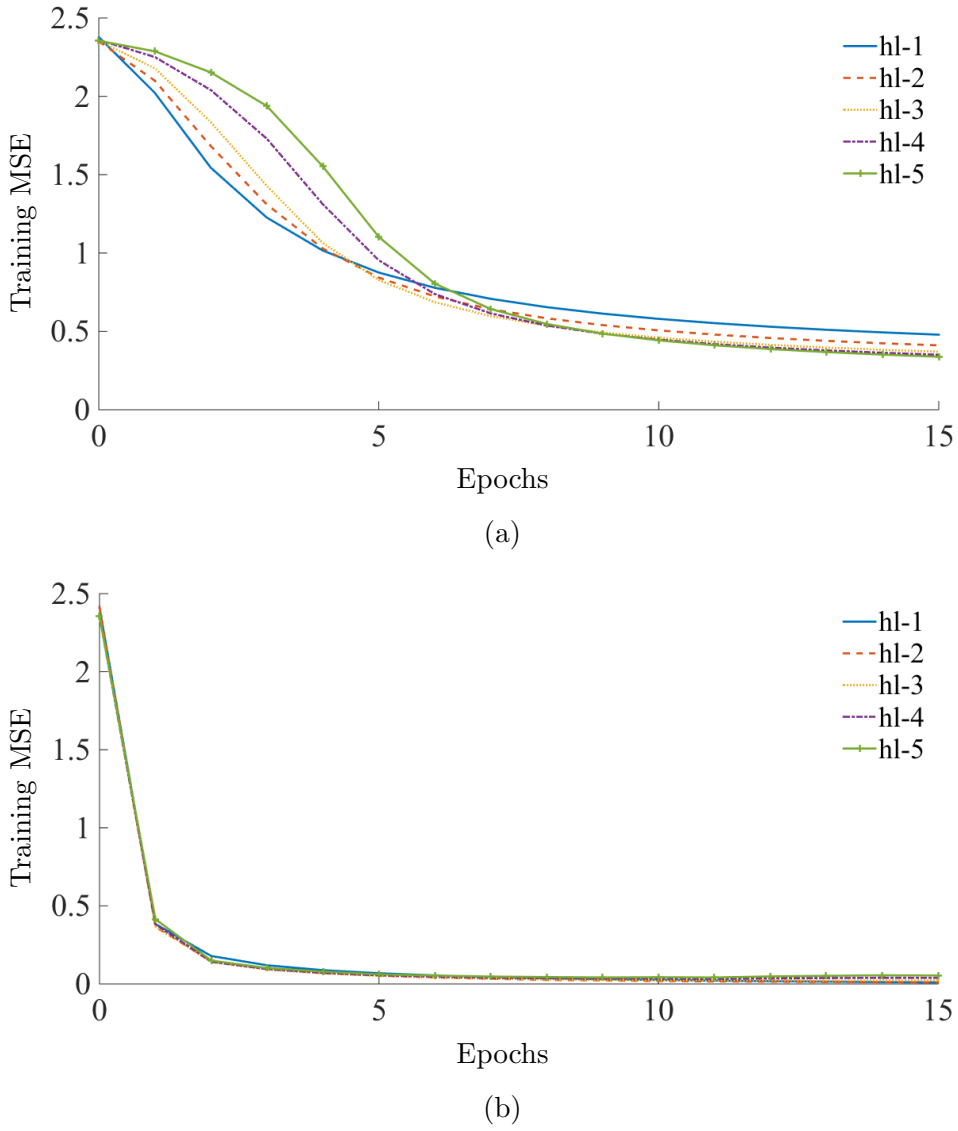

Figure 4.6: Training mean squared error of varying number of hidden layers with ReLU activation function. (a) Training with SGD and (b) training with SDAGD.

slower training because of fixed learning rate utilizing similar step size across the entire training. SDAGD algorithm on the other hand has iteration steps adaptively tuned based on the local search regions as shown in Figure 4.6(b). Hence, relative step length with long-term optimal trajectory adaptation is able to provide a smoother training curves. Applying SDAGD algorithm with ReLU activation still outperformed SGD algorithm while not showing any sign of vanishing gradient throughout the experiment.

Table 4.1 tabulates the misclassification rate (MCR) of SGD and SDAGD algorithms. SDAGD algorithm with ReLU activation function outperformed SGD with the best miscalssification rate at 1.77% on hl-1 configuration. Conversely, both SGD and SDAGD algorithms observed monotonic increase in MCR in response to the increasing number of hidden layers. This phenomena implies that the network is getting harder to train with higher level of abstraction. However, SGD with sigmoid activation function encountered problem with training as the number of hidden layers increases. The growing of MCR as the number of hidden layer increases shows sign of vanishing gradient. The entire training process is halted with hl-4 and hl-5 configurations. In comparison, SGD is still able to obtain decent result in ReLU configuration as ReLU is designed to resolve vanishing gradient problem in deep neural networks. SDAGD algorithm works consistently in both sigmoid and ReLU activation functions, demonstrating that SDAGD algorithm is free from vanishing gradient problem.

Table 4.1: Testing misclassification rate for SGD and SDAGD with ReLU and sigmoid activation functions.

Configuration	SGD		SDAGD	
	Sigmoid	ReLU	Sigmoid	ReLU
hl-1	10.73	10.95	3.34	1.77
hl-2	12.98	10.07	3.66	1.96
hl-3	33.71	9.32	4.00	2.09
hl-4	88.65	9.24	4.11	2.09
hl-5	89.91	9.22	5.07	2.34

4.5 Chapter Summary

The Stochastic Diagonal Approximate Greatest Descent (SDAGD) is tested using three set of benchmark error surfaces, i.e.: (a) a hilly error surface with two local minima and one global minimum; (b) a deep Gaussian trench to simulate drastic gradient changes experienced with ravine topography and (c) small initial gradient to simulate a plateau terrain. Based on optimization visualization, SDAGD demonstrated better converging trajectory in all the three simulated problems when compared to SGD,

AdaGrad and AdaDelta. SDAGD algorithm possessed the ability to compute different relative step-length adaptively to seek for the global minimum. The Hessian element in SDAGD algorithm describes the local curvature information into optimization to avoid trapping in plateau and overrunning ravine. Additionally, the spherical search regions constructed by the relative step-length helps to keep the trajectory rolling, thus mitigating the vanishing gradient problem in gradient-based learning method. Current practice to overcome this issue is to increase the variants of neural network architecture by replacing the saturated activation function such as sigmoid function with the unsaturated activation function such as Rectified Linear Unit (ReLU). However, unsaturated activation function may lead to exploding gradient issue if the hyperparameters are not properly tuned. In the experiments, MLP structure sequentially adding layer by layer is tested with the proposed SDAGD to study the effects of vanishing gradient using saturated and unsaturated activation functions. The experiments showed that SDAGD can overcome the saturated activation function's vanishing gradient issue as compared to SGD. It can further reduce the MCR to 2.34% by using unsaturated activation function without exploding gradient issue. This result concludes that SDAGD can mitigate the vanishing gradient by avoiding error backpropagation in smaller gradient due to the adaptive learning rate element. The proposed SDAGD will be explored using deeper convolutional neural networks with large scale database in the next chapter.

Chapter 5

Deep Convolutional Neural Networks using Stochastic Diagonal Approximate Greatest Descent

5.1 Introduction

Convolutional Neural Networks (CNNs) are the state-of-the-art in machine learning for data processing, such as images, videos, and speech [82]. As compared to the conventional machine learning techniques, CNN can process natural data by extracting and learning data feature representation itself through the network learning scheme. In a practical perspective, CNN requires plenty of labeled data to train the weight for pattern recognition. CNN consists of an input layer, multiple successive convolutional layers and pooling layers, and fully-connected layer. A convolutional layer consists of multiple kernels/filters to map the input to the next layer dubbed as feature maps. The feature maps utilize weights sharing therefore each filters is subjected to detect a specific patterns, edges or motifs. A pooling layer combines a few semantically related features from the feature maps of the previous layer. The common methodology used in pooling are average pooling and max pooling. Subsequently, a fully-connected layer works just like the feed-forward network to classify the obtained features at the last few

layers. The breakthrough of CNN from vanilla neural networks is due to the adaptation of global receptive fields, shared weights and spatial or temporal subsampling. Each element provides better conversion of natural image into higher level of abstraction for better inference on the test objects.

Some well established CNN models that are essential in DNN advancements include LeNet, ZF Net, GoogLeNet, VGG Net and ResNet. Each of the network possesses its strength and weaknesses to deal with different computer vision problems. LeNet [44] pioneers the application of CNN in the field of computer vision. LeNet-5 is a 5 layer CNN with an input layer, 3 convolutional layer followed by pooling layer and a fully-connected layer as shown in Figure 5.1. The notable application of LeNet-5 is trained on MNIST dataset to read bank cheques. Since it is the first form of CNN, most of the parameters are not optimized to its peak performance and are subjected to vanishing gradient problem. AlexNet [6] sets the trends for the application of convolutional neural network in computer vision due to it's overwhelming recognition rate. AlexNet is the winner of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 with the lowest recognition error. The structure of the network is similar to that of LeNet but in a larger and deeper manner. AlexNet also utilizes unsaturated activation function, rectified linear unit (ReLU) instead of sigmoid or tanh function. The fact that AlexNet is larger and deeper, it is more computationally expensive than LeNet and are usually trained with GPUs.

ZF net [90] is the winner for ILSVRC 2013. ZF net leveraged from AlexNet with better hyperparameter fine-tuning. The authors cover the intuition behind a good performance CNN particularly on how to optimize the filter and weights to improve the performance. The tweak is made on the stride and filter size of the first layer and the network size of the middle part of ZF net. A smaller choice of stride and filter size used in ZF net allows retaining of original pixel information at the beginning of the network. The size of stride and filters increases as the network grows larger to further coarse-grain the input signal into a higher level of representation. ZF net also utilizes more filters and therefore the training time is generally longer than AlexNet. The authors also developed a visualization technique dubbed as deconvolutional network to look at the high level features alongside with training. Deconvolutional network helps to examine the training specific to certain desired features excites by the input.

GoogLeNet [91] was the winner at ILSVRC 2014. GoogLeNet introduced inception module to reduce the number of parameters in the network. Instead of stacking plain

convolutional layers with pooling layers, GoogLeNet is constructed via a stack of inception modules. Each inception module performs 1×1, 3×3, 5×5 convolutional and 3×3 max pooling in parallel. Despite the fact that each inception module consists of different sizes of filters, the features extracted is ranged from fine grain to coarse abstraction of the input signal. Every pieces of information from the input signal are fully utilized and thus it outperformed the other CNNs. Additionally, it utilized average pooling for the final fully connected network to reduce the total number of parameters. However, GoogLeNet is an extremely large and deep network that requires massive memory cost and computing power in return for better performance.

VGGNet [7] scores the second place in ILSVRC 2014. VGGNet utilized only 3×3 convolutional filter and 2×2 pooling layer for the entire CNN. The authors used 2 smaller filters sizes to achieve somewhat similar results of a single larger filters. It again showed the benefits of having deeper convolution model for image classification task. The goal of VGGNet is to keep the optimization of network hyperparameter at minimum by increasing the depth of the network. Although it is much simpler to be executed, the downside of VGGNet is that it requires more memory as the depth of the network grows deeper.

Residual network (ResNet) [8] was the winner for ILSVRC 2015. It is the current state-of-the-art for CNN in the field of computer vision. ResNet set the deepest CNN with 152 layers of stacking convolutional layer. ResNet is made up of residual block. The residual block consists of 3×3 convolutional layers and 2×2 max pooling layer. Each residual block consists of convolutional-ReLU-convolutional series. Apart from conventional forward pass, the output of the residual block is added with the input. Basically, this operation computes the delta or small change to the original input signal. Conversely, a traditional CNNs will only have the abstract features and has nothing to do with the input anymore. The residual block is able to retain as much information as possible as the network grows deeper. The down side of an ultra-deep network increased computations and memory required to complete the task.

Most of the structural changes in CNN is associated with the conventional optimization problem such as vanishing gradient. ResNet in particular is an example that utilizes residual signal to propagate higher amplitude of information across the network to prevent vanishing gradient. This is due to the utilization of gradient-based approach in CNN's backpropagation. The drawbacks of using gradient-based method are vanishing gradient, learning rate dependency and susceptible to be trapped in local minima.

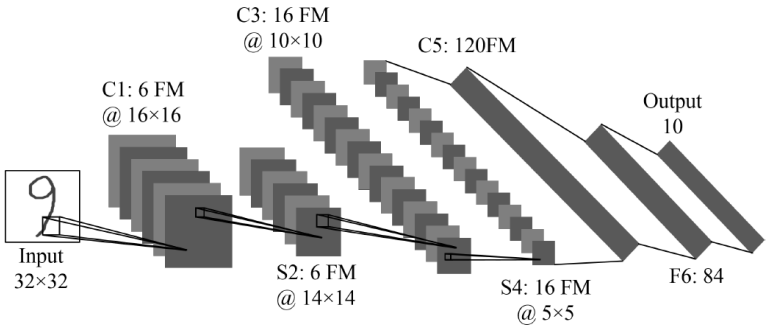

Figure 5.1: Architecture of convolutional neural networks (LeNet-5).

Moreover, the learning process of gradient-based approach also requires more numerical iterations and training samples to achieve better results. Hence, SDAGD algorithm is proposed to further improve the learning capability and structural simplicity of CNN. SDAGD algorithm involves local curvature information in iteration steps construction that are improving the learning capability of CNN. This is due to the ability of determining more optimal step direction at every iteration towards the minima point. A more optimal step direction especially when dealing with ultra deep neural networks could help to prevent divergent from occurring.

In this chapter, an overview of multiple variances of convolutional neural network is covered. Subsequently, the theory and formulation of error backpropagation of CNN are presented in Section 5.2. Section 5.3, 5.4 and 5.5 cover a few image classification experiments of deep CNN with multiple complex datasets. Lastly, the chapter's recap is summarized in Section 5.6.

5.2 General Theory of Convolutional Neural Network

Convolutional Neural Network involves convolutional layers, pooling layers and fully connected layers. Convolutional layer consists of multiple kernels/filters that map the previous layer through weights sharing and produce an output called feature maps. Each feature map has its unique set of weights parameter to construct different edges and motifs. The convolution function can be defined as,

$$Y^{(l)} = W_{i,j}^{(l)} * X_{i,j}^{(l)}, \tag{5.1}$$

where $Y^{(l)}$ is the output vector, $W_{i,j}^{(l)}$ is weight vector defined in $C_x \times C_y$ filter size, $X_{i,j}^{(l)}$ is the input vector, l represents l-th layer and $*$ defines convolution function as,

$$Y^{(l)} = \sum_{i'} \sum_{j'} W_{i',j'}^{(l)} X_{i-i',j-j'}^{(l)}. \tag{5.2}$$

The weighted sum is then passed to an activation function to perform non-linearity transformation. A pooling layer combines a few semantically related features to coarse-grain the weighted sum features. Pooling layer provides translation invariance properties so that recognition is not subjected to shifting or rotation. Pooling layers only perform subsampling and no learning process is involved in this layer. Average pooling can be written as follows,

$$g\big(Y^{(l)}\big) = \frac{\sum_{i=1}^{m} y_i^{(l)}}{m}, \tag{5.3}$$

where $g(\cdot)$ represents average pooling function, m is the total number of output elements in l-th layer. Fully connected layer behaves like ordinary neural networks that are utilized for classification tasks. Many layers of convolutional layer and pooling are stacked up followed by fully connected layer as last layer to form a CNN. Output layer is denoted as:

$$X^{(L)} = F(Y^{(L-1)}) = F(W^{(L-1)} X^{(L-1)}), \tag{5.4}$$

where $X^{(L)}$ represents the output vector of layer L computed from weighted sums of input vector $Y^{(L-1)}$ with $F(\cdot)$ as activation function.

Error signal computed based on objective function are backpropagated through CNN for weights updating. The objective function can be written in an incremental quadratic manner as follows,

$$E = \frac{1}{2}(D - X^{(l)})^2, \tag{5.5}$$

where X is the output vector, D is the target vector and L denotes the last output layer. The learning process is repeated until the error drops to a small margin. Multiple layers of convolution and pooling are stacked up with the output of each layer acted as the input of the adjacent layer. The error signal at the last fully connected layer is computed via partial derivatives of objective function with respect to weights as,

$$\frac{\partial E}{\partial W^{(l)}} = \frac{\partial E}{\partial Y^{(L)}} \frac{\partial Y^{(L)}}{\partial W^{(L)}} = \frac{\partial E}{\partial Y^{(L)}} X^{(L-1)}, \tag{5.6}$$

$$\frac{\partial E}{\partial Y^{(l)}} = F'(Y^{(l)}) \frac{\partial E}{\partial X^{(L)}}, \tag{5.7}$$

where $F'(\cdot)$ is the derivatives of $F(\cdot)$. Backpropagation in pooling layer is much simpler compared to fully connected layer because no learning is involved. The pooling layer simply forward the error signal from next layer to previous layer for update. Thus, the error signal can be written as,

$$\frac{\partial g(Y^{(L)})}{\partial Y^{(L)}} = \frac{1}{m}, \tag{5.8}$$

where m is the total number of output elements in l-th layer. As for the convolutional layer, error signal can be computed by applying chain rules to convolution formula shown in the forward pass. Hence, it can be written as,

$$\frac{\partial E}{\partial W_{i,j}^{(l)}} = \sum_{i'} \sum_{j'} \frac{\partial E}{\partial y_{i',j'}^{(l)}} \frac{\partial y_{i',j'}^{(l)}}{\partial W_{i,j}^{(l)}}, \tag{5.9}$$

$$\frac{\partial y_{i',j'}^{(l)}}{\partial W_{i,j}^{(l)}} = \sum_{i'} \sum_{j'} \frac{\partial}{\partial W_{i,j}^{(l)}} \Big(\sum_{i''} \sum_{j''} W_{i'',j''}^{(l)} X_{i'-i'',j'-j''}^{(l)} \Big). \tag{5.10}$$

Since the partial derivatives of components in (5.10) result in zero except when $i'' = i$ and $j'' = j$, it can be simplified as,

$$\frac{\partial y_{i',j'}^{(l)}}{\partial W_{i,j}^{(l)}} = \frac{\partial}{\partial W_{i,j}^{(l)}} \Big(W_{i,j}^{(l)} X_{i'-i,j'-j}^{(l)} \Big) = X_{i'-i,j'-j}^{(l)}. \tag{5.11}$$

After substituting back into the error equation, yields,

$$\frac{\partial E}{\partial W_{i,j}^{(l)}} = \sum_{i'} \sum_{j'} \frac{\partial E}{\partial y_{i',j'}^{(l)}} X_{i'-i,j'-j}^{(l)}. \tag{5.12}$$

5.3 Experiment #1: LeNet-5 with MNIST dataset

The Mixed National Institute of Standards and Technology (MNIST) database [44] was used to verify the performance of SDAGD algorithm with CNN. MNIST database contained $60,000$ examples as training set and $10,000$ examples as testing set. All digits were size normalized and centered in a dimensions of 28×28 square image. $5,000$ examples were chosen equally across all digits (500 examples from each digit) and used as a training set. As for testing set, the full $10,000$ testing images will be used. Sigmoid function was chosen as the activation function to train with SDAGD algorithm in LeNet-5 configuration for prove of vanishing gradient mitigation.

Three experiments of SGD with different learning rate is constructed to demonstrate the needs of fine-tuning process. A properly tuned SGD learning algorithm achieves a

better MCR rate at 13.68% compared to SDLM at 15.59%. However, Hessian-based approach could provide more reliable results without the needs to repeat the training multiple times. This is because Hessian-based approach utilizes the local curvature information in weights optimization. Additionally, SDAGD algorithm achieves the best overall MCR at 8.58%. This result demonstrated the utilization of long-term optimal trajectory in neural networks weight updating scheme. Iteration steps were constructed in more optimized manner and therefore yielded better results as reported in Table 5.1.

Table 5.1: Training and testing misclassification rate using LeNet-5.

Training Algorithm	**Configuration**	**MCR(%)**	
		Training	**Testing**
SGD	$\eta = 0.02$	46.04	46.08
	$\eta = 0.002$	13.56	15.21
	$\eta = 0.0002$	13.86	13.68
SDLM	$\eta = 0.0002, \mu = 0.02$	14.16	15.59
SDAGD	$R = 0.1$	5.72	8.58

On the other hand, the MSE of SGD, SDLM and SDAGD algorithms are illustrated in Figure 5.2. SDAGD algorithm is able to achieve lower training error at faster roll-off speed as compared to other learning algorithms. At the third epoch, SDAGD algorithm arrived at the lowest error rate. As a result, SDAGD algorithm possesses the advantage from the effect of long-term optimal trajectory in iteration construction. Relative step-length in AGD provided better capability to approximate step length between iteration steps. Hence, it is observed that SDAGD algorithm achieved lower MSE faster by adaptively taking larger step length when solution is far away. Subsequently, relative step length is gradually reduced when it is nearer to the solution. Both SGD and SDLM were only able to achieve MSE of around 1.1 whereas SDAGD algorithm achieved 0.8 of MSE as shown in Figure 5.2.

SDAGD algorithm also benefits from the implementation of stochastic diagonal training by having better network generalizability and lower computational cost. Experiment result showed that SDAGD algorithm achieved 8.58% of misclassification rate on MNIST database. Moreover, SDAGD algorithm also offers faster roll-off during learning phase than other existing learning algorithm. The results show the benefits of adaptive

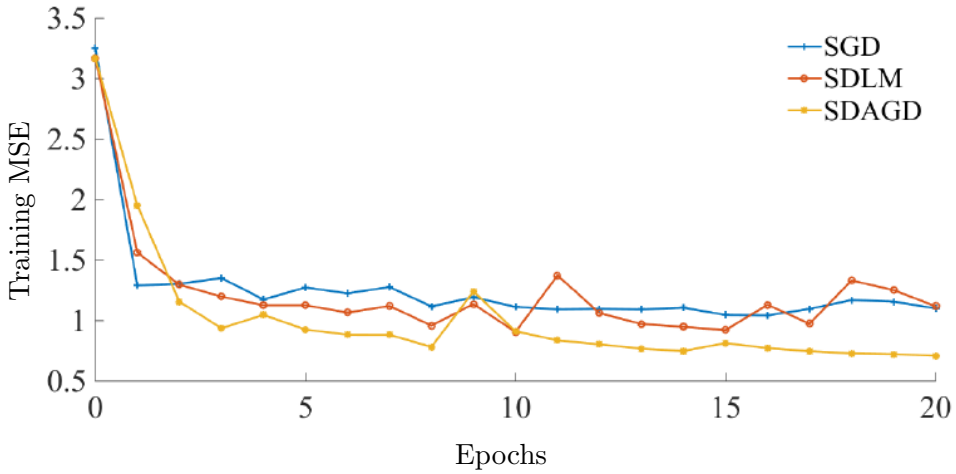

Figure 5.2: Training mean squared error with respect to epoch using LeNet-5.

Hessian-based optimization system in the construction of more optimal iteration steps for neural network training.

5.4 Experiment #2: AlexNet with CIFAR-10 dataset

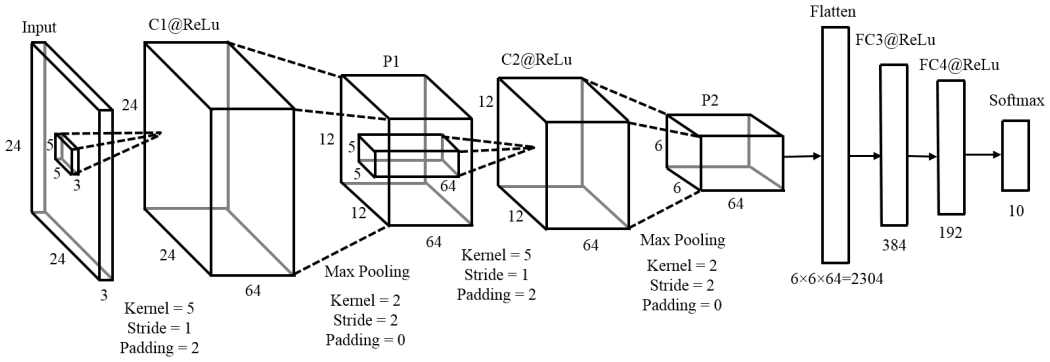

Figure 5.3: Architecture of modified AlexNet with three dimensional inputs.

A CNN modified from AlexNet [6] is setup as depicted in Figure 5.3 to run on CIFAR-10 [45] dataset. Each convolutional layer will performs convolution, local response normalization and ReLU [88] activation function. CIFAR [45] dataset as shown in Figure 5.4 is more complex than MNIST dataset because it includes the colour spectrum, i.e. red, green and blue. The CIFAR-10 dataset consists of 60,000 32×32 colour images distributed into 10 classes. Each class contains 6,000 images. CIFAR-10 is split

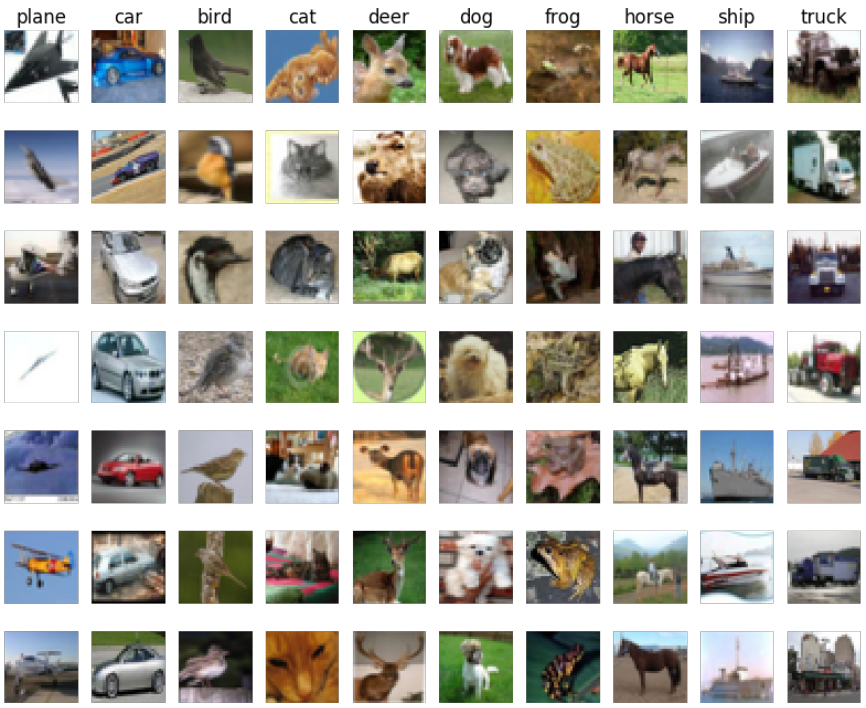

Figure 5.4: Example of CIFAR-10 images with 10 output classes.

into 50,000 training images and 10,000 test images. Each training set is tied with an output label for weights optimization. The 10 classes of objects include airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. For experimental purposes, each training images were cropped to dimension of 24×24 with approximate whitened and were randomly distorted to increase the training size. Image padding is inserted after convolution process. The constructed network is trained with mini-batch SGD, ADAM and SDAGD algorithms with batch size of 128 inputs per batch for performance comparison.

Figure 5.5 depicts the training curve for ADAM, SDAGD and SGD algorithms. ADAM and SDAGD algorithms possess faster roll-off rate as compared to SGD algorithm. SGD algorithm struggles from slower training because of fixed learning rate that utilizes similar step size across the entire training. In SDAGD algorithm, iteration steps are adaptively tuned based on the local search regions and hence the relative step length with long-term optimal trajectory adaptation is able to contribute to a better estimation of step-length. This adaptive step-length estimation outperformed SGD algorithm and is comparable to ADAM algorithm.

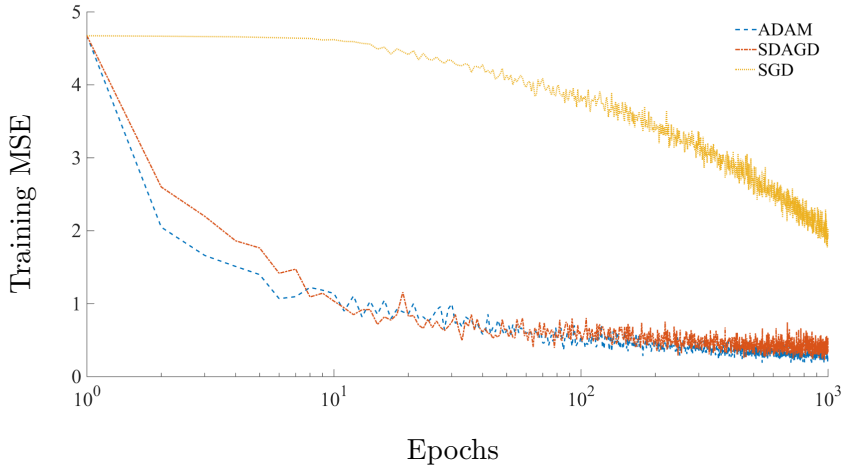

Figure 5.5: Training mean squared error with respect to epoch using modified AlexNet.

Table 5.2: Testing misclassification rate of SGD, ADAM and SDAGD optimizers using modified AlexNet.

Learning algorithm	Configuration	MCR (%)
SGD	$\eta = 0.0001$	27.5
ADAM	$\eta = 0.0001$	13.6
SDAGD	$R = 0.1$	13.8

Table 5.2 tabulates the misclassification rate of ADAM, SDAGD and SGD algorithms. As a result, SDAGD algorithm achieved comparable MCR to ADAM training algorithm. For comparison purposes, all the learning rates are fixed at 0.0001. However, a properly tuned SGD could still achieve decent results with longer training iterations. In this experiment, SDAGD algorithm achieved MCR of 13.8% which are comparable to ADAM at 13.6%. SDAGD algorithm also obtained a better training curve comparable to ADAM algorithm. This is achieved through the adaptation of relative step length in SDAGD algorithm enabling more adaptive step-length construction throughout the optimization process.

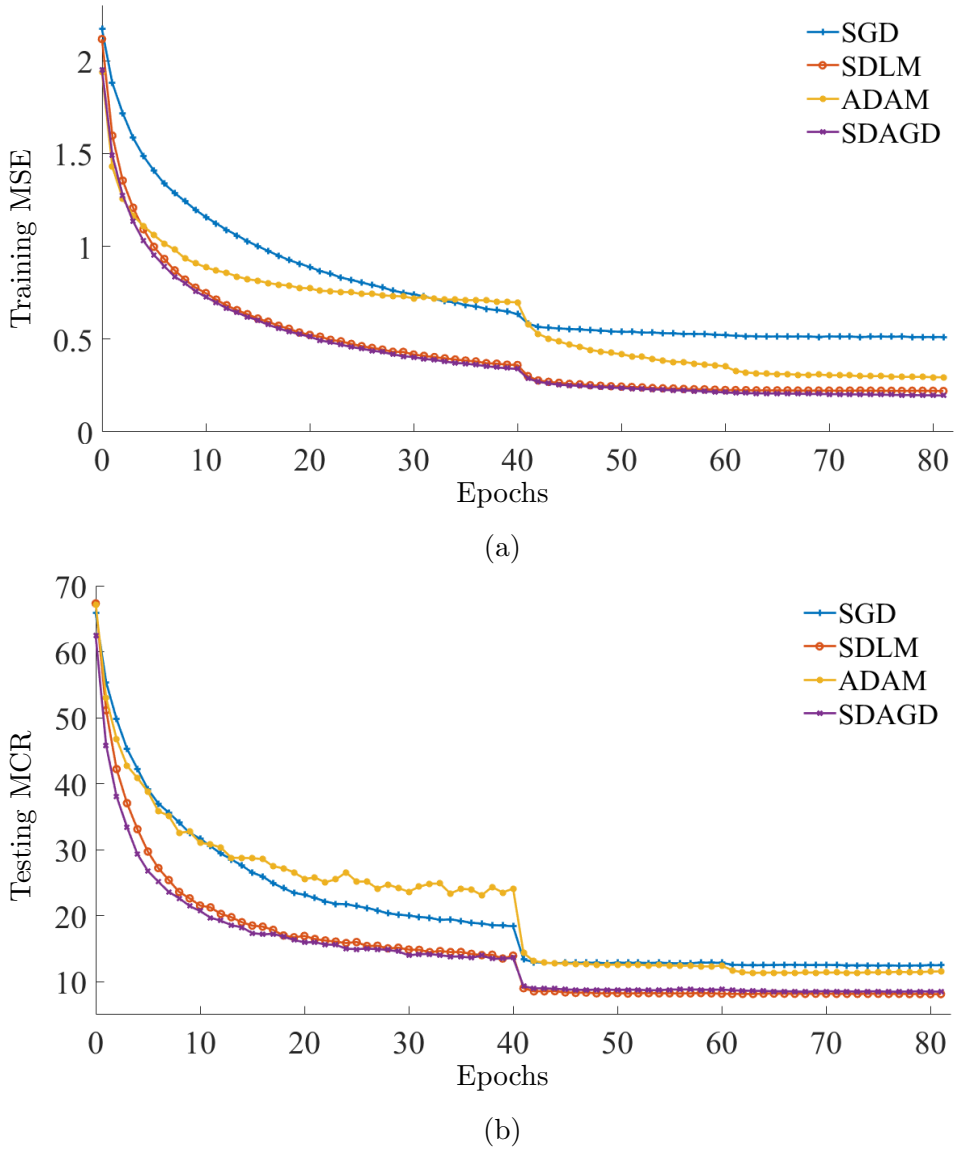

Figure 5.6: The training of ResNet-34 with CIFAR-10 dataset. (a) Training MSE curve and (b) Testing MCR curve.

5.5 Experiment #3: ResNet-34 with CIFAR dataset

This experiment is designed to demonstrate the efficacy of the proposed method with large-scale image classification. The experiments are setup based on the residual network (ResNet) [8] with CIFAR-10 and CIFAR-100 dataset. The dataset contains $50,000$ 32×32 RGB images in the training set and $10,000$ images in the testing set. CIFAR-100 is distributed into 100 output classes containing 600 images each with 500

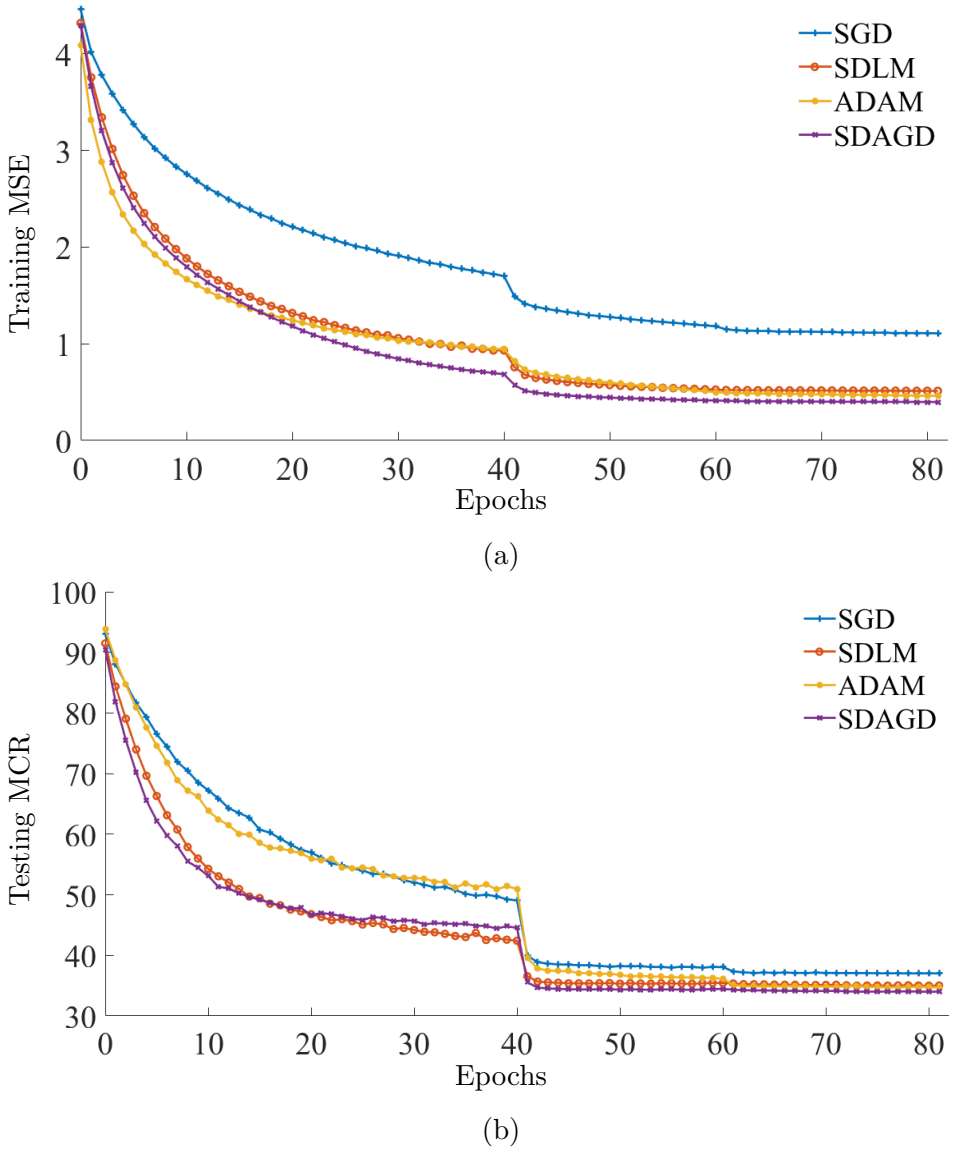

Figure 5.7: The training of ResNet-34 with CIFAR-100 dataset. (a) Training MSE curve and (b) Testing MCR curve.

training images and 100 testing images per class, while CIFAR-10 dataset contains 10 output classes. The chosen network is ResNet-34 architecture as shown in Figure 5.8. ResNet are made up of residual block consists of 3×3 convolutional layers and 2×2 max pooling layer. Each residual block is made up of convolutional-ReLU-convolutional series. The experiments are setup to compare the proposed method with SGD, SDLM and ADAM. SGD, SDLM and ADAM algorithms will have the starting learning rate set

at 0.1, 0.001 and 0.01 respectively while the radius hyperparameter of SDAGD algorithm is set at 0.1 for best performance. All the networks are trained for a total 80 epochs and reduced the learning rate by factor of 10 on epochs 40 and 60.

The results demonstrate the efficacy of the proposed method in DNN with CIFAR-10 and CIFAR-100 datasets. Based on Figure 5.6(a) and Figure 5.7(a), the training curve of SDAGD algorithm descends faster than ADAM. The difference in performance lie with the two-phase switching optimization approach of SDAGD algorithm. SDAGD takes smaller steps near to the end of training to venture into different basin of error surfaces. SGD algorithm descends at the lowest and a properly tuned SDLM algorithm acquires similar performance to SDAGD algorithm. The testing MCR as depicted in Figure 5.7(b) and Figure 5.6(b) demonstrate the close resemblance of testing curve to training curve indicate that the network is not over-fitted. Table 5.3 tabulates the MCR of SGD, SDLM, ADAM and SDAGD algorithms. The extra parameters of SDLM algorithm requires fine-tuning to achieve optimal results. The lower MCR by SDAGD algorithm indicates that the optimizer is able to explore long-term trajectory to reach the optimal solution in an error space. As a conclusion, SDAGD using two-phase switching strategy has the advantages of steepest roll-off in training phase, adaptive adjustment in the relative step-length using Hessian-based error information and the ability to deal with vanishing gradient issues in deep learning neural networks.

Table 5.3: Testing MCR for ResNet-34 with CIFAR dataset.

Training Algorithm	Configuration	MCR(%)	
		CIFAR-10	CIFAR-100
SGD	$\eta = 0.1$	11.93	37.57
SDLM	$\eta = 0.001$, $\mu = 0.1$	31.94	75.45
	$\eta = 0.001$, $\mu = 0.01$	11.1	37.38
	$\eta = 0.001$, $\mu = 0.001$	7.6	36.72
ADAM	$\eta = 0.01$	11.11	35.67
SDAGD	$R = 0.1$	7.98	33.62

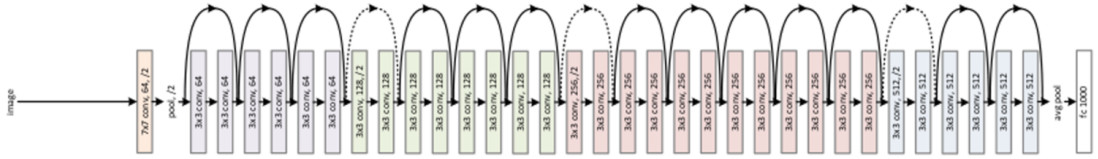

Figure 5.8: Architecture of ResNet-34 convolutional neural network.

5.6 Chapter Summary

Stochastic Diagonal Approximate Greatest Descent (SDAGD) is applied to deep convolutional neural network (CNN) using large-scale dataset as test cases to evaluate the capability with large-scale optimization problem. Three recent deep architecture of neural network, i.e. LeNet-5, AlexNet and ResNet-34 are tested with MNIST, CIFAR-10 and CIFAR-100 respectively to observe training error and testing recognition rate. LeNet-5 is the pioneer application of CNN in computer vision while AlexNet outperforms the usage of CNN in image classification using ADAM optimizer. As the datasets are large and complex, increasing depth of CNN architecture has become the practice in the research of machine learning. ResNet-34 improves standard deep CNN architecture by adding residual blocks in the forward propagating structural to solve the vanishing gradient issue. From the experiment of AlexNet with CIFAR-10 dataset, SDAGD algorithm achieves a misclassification rate of 13.8% which is comparable to ADAM algorithm with 13.6%. As for large-scale image classification task, SDAGD is utilized in ResNet-34 with CIFAR-10 and CIFAR-100 dataset. SDAGD with annealing radius on ResNet-34 with CIFAR-10 dataset achieves 7.98% of misclassification rate when compared to ADAM at 11.11%. As for ResNET-34 with CIFAR-100 dataset, SDAGD algorithm achieves misclassification rate of 33.62%, which is lower than ADAM optimizer at 35.67%. As a conclusion, SDAGD with annealing radius algorithm is able to provide consistent steeper training curve than other methods with higher recognition rate by using the adaptive learning rate derived based on long-term optimal control theory.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

An adaptive second-order derivative optimization framework was presented with stochastic diagonal approximate greatest descent (SDAGD) algorithm to optimize artificial neural network. SDAGD is derived from the operation of a multi-stage decision control system. The control system is generally a two-phase approach where: (a) when the local search region does not contain a minimum point, the iteration shall be defined at the boundary of the local search region; (b) when the local region contains a minimum point, the approximate Newton method is used to search for the optimum solution. The implementation of SDAGD algorithm in MLP and CNN are investigated with the goal of improving the learning ability while retaining structural simplicity. From all experiments conducted, the results demonstrate that SDAGD algorithm is efficient in both shallow and deep artificial neural network training, achieve better recognition rate when compared with other existing training algorithms, suitable for most networks as the implementation is similar to gradient-based approach and requires no extra memory to store previous gradients information for optimization.

The learning curve of artificial neural network is one of the well-studied area in machine learning. The main implication from the literature review is to identify the effective and robust optimization technique for artificial neural network training. Chapter 2 covers an extensive review on the current state-of-the-art optimization techniques used in the field of DNN learning. The optimization techniques related to this work are generally segregated into three categories: gradient-based approach, Hessian-based approach and adaptive learning rate method. This chapter also pinpoints the problems

with the existing optimization techniques followed by discussion of possible or existing solutions. This chapter emphasizes the missing relation between adaptive approach with Hessian-based approach which is then proven to provide better performance in the experiments.

In Chapter 3, the formulation of proposed stochastic diagonal approximate greatest descent (SDAGD) algorithm is derived with two approximations: applying truncated Hessian and omitting the off-diagonal terms. SDAGD algorithm is computationally efficient and free from memory dependency to store previous Hessian information. Eventually, SDAGD algorithm inherits the concept of long-term optimal trajectory control theory that is excel in formulating an optimal trajectory towards the solution. SDAGD algorithm is proven convergence based on the Lyapunov function theorem supported by the numerical experiment showing parameters subsiding to small deltas with respect to increasing training iterations. In the experiment, a two-layer shallow multilayer perceptron (MLP) is built to test on the SGD, SDLM and SDAGD optimizers using MNIST dataset. Due to the adaptive learning using Hessian-based approach, the steep descend in the learning weights and errors of SDAGD can be observed during the training process. The proposed SDAGD achieves misclassification rate of 3.34% as compared to SGD (3.81%) and SDLM (4.00%). The experiment also defines the radial effect of SDAGD annealing to provide better misclassification rate performance of 1.62%. By annealing the radius of the spherical search regions, SDAGD performs normalization to scale up the relative step length to explore optimal solution at the optimization level set of phase-II.

Chapter 4 presents the optimization visualization of SDAGD algorithm. SDAGD algorithm demonstrates robust converging trajectory with three different simulated topographies, i.e.: (a) a hilly error surface with two local minima and one global minimum; (b) a deep Gaussian trench to simulate drastic gradient changes experienced with ravine topography and (c) small initial gradient to simulate a plateau terrain. SDAGD also demonstrates adaptive learning rate element with the effects of Hessian-based information, which is capable to deal with ravine and plateau topographies and converge to the expected solution. On the other hand, gradient-based learning method always encounters the issue of vanishing gradient in neural network backpropagation causing the weight training process to halt without reaching to the final solution. SDAGD addresses the vanishing gradient problem in deep neural network. SDAGD algorithm demonstrates the ability to mitigate the issue of vanishing gradient in deep feedforward

network without sign of early training termination. The results show that SDAGD is able to obtain good performance in the deep feedforward network; while SGD obtains the worst misclassification error from three to five hidden layers. This result concludes that SDAGD can mitigate the vanishing gradient by avoiding error backpropagation in smaller gradient due to the adaptive learning rate element. SDAGD achieves significantly lower misclassification rate of 2.34% as compared to properly tuned SGD at 9.22% using MNIST dataset.

As for large scale image classification problem, the capability of SDAGD algorithm is explored in Chapter 5 on more sophisticated image classification problem with deeper network. This chapter covers the used of SDAGD algorithm in convolutional neural network designed to test on complex and large dataset for image classification. LeNet-5, AlexNet and ResNet-34 are tested with MNIST, CIFAR-10 and CIFAR-100 respectively to evaluate the robustness of SDAGD algorithm. From the experiment of AlexNet with CIFAR-10 dataset, SDAGD algorithm achieves a misclassification rate of 13.8% which is comparable to ADAM algorithm with 13.6%. As for the experiment with ResNet-34 on CIFAR-10 and CIFAR-100 dataset, SDAGD with annealing radius achieves lower misclassification rate (7.98% for CIFAR-100 and 33.53% for CIFAR-100 dataset) when compared to SGD and ADAM optimizers. As a conclusion, SDAGD with annealing radius algorithm is able to provide consistent steeper training curve than other methods with higher recognition rate by using the adaptive learning rate derived based on long-term optimal control theory. SDAGD using two-phase switching strategy has the advantages of steepest roll-off in training phase, adaptive adjustment in the relative step-length using second-order derivative error surface and the ability to deal with vanishing gradient issues in deep learning neural networks.

6.2 Future Work

The limitation of computational efficiency of SDAGD algorithm can be improved with newly implemented long-term optimal optimization technique. Alternative to AGD, the bang-bang iteration [65], which shares similar backbone of AGD emerges recently with rectangular search regions could essentially explore other benefits of utilizing two-phase approach in DNN optimization. The bang-bang iteration is a simpler two-phase approach. During phase-I, the rectangular search regions formulated from

the bang-bang iterations is constructed. The bang-bang iterations are useful and cost-effective because it only involves gradient information for each iteration. In phase-II, the original AGD method with spherical search regions is then used for quadratic convergence. This replacement is done because the AGD method with spherical search regions is able to handle the singularity of the Hessian matrix along the iterations. The author also covered a sub-space approach to handle large-scale optimization problems. The sub-space approach involves a lower dimension of the Hessian matrix while proving the robustness but mitigating the needs of expansive computation and less memory intensive than current approaches. Despite having bang-bang iteration vastly similar to the proposed method theoretically, there will still be challenges incorporating the bang-bang iteration as well as the sub-space approach into neural network optimization. If done properly, the effectiveness of the proposed method is anticipated to increase by having reduced computational cost during phase-I, and less memory consumption for phase-II with the sub-space approach.

For the future development, SDAGD algorithm can be applied into ultra deep network e.g. ResNet-156 with ImageNet dataset to further analyse the latent capability of SDAGD algorithm. When the network grows deeper, the signals and features become minuscule and causes over saturation on the first few layers in the higher hierarchical order. The problem is twofold especially with the conventional optimization techniques without adaptive, involving Hessian element and two-phase approach. The deeper layer no longer contributes to better inference, while find spending more time to learn that smaller network achieved similar results as compared to vanished deeper network. The needs to incorporate regularization into training or changing the structure of the network mildly might solve the issues, but in return added complexity to the overall network design as the same time. All in all, the added complexity contributes more hyperparamters for fine-tuning besides the already troublesome hyperparameters fine-tuning needed for optimization. The current dilemma is that most recent data and images are complex by nature, the essential needs for deep learning is definitely growing. Based on the experiment, SDAGD algorithm demonstrated positive behavior in terms of vanishing gradient problem mitigation and the ability to adaptively switches step length based on the topography of the error surface. However, the response on ultra deep network is still yet to be discovered.

In addition, SDAGD algorithm can be applied into natural language processing (NLP), which involves recurrent neural network (RNN). Traditional artificial neural

networks secure no connections on the previous events, the dataset are not temporally related and each data point is mostly confined to one output. Traditional artificial neural networks are only connected in a top-down manner, which only involve interconnection between layers and only suitable to classify one-off event rather than event with historical influences. As for natural language processing or any other temporal dataset, RNN is designed to have intraconnection between layer. In the structural context, a smaller network's output is cascaded to next series of smaller networks. Hence, the classification on the final output is dependent to the previous output, followed by the previous output until the first input. In contrast, traditional neural networks only take fixed sized vector as input and produce fixed sized vector as output. However in NLP, the input and output size is not predetermined, and is subjected to change based on the input and is pseudo-random in a sense that repeated input will not provide similar output. For example, the sentences predicted for a translation task might change in terms of vocabulary and word choices, but the meaning stays the same. The structural different contributes to difficulties in RNN backpropagation, usually called the backpropagation through time. It involves previous input as part of the error signal backpropagated through the network. Due to higher complexity with backpropagation through time, the effect of SDAGD algorithm with neural networks that are confined temporally is yet to be explored.

Bibliography

[1] L. Deng, "Three classes of deep learning architectures and their applications: a tutorial survey," *Transactions on Signal and Information Processing*, 2012.

[2] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends ® in Signal Processing*, vol. 7, no. 34, pp. 197–387, 2014.

[3] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65(6):386, 1958.

[4] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, July 2006.

[5] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML'09. New York, NY, USA: ACM, 2009, pp. 609–616.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105.

[7] D. Yu, S. Wang, Z. Karam, and L. Deng, "Language recognition using deep-structured conditional random fields," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, March 2010, pp. 5030–5033.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] J. Tompson, A. Jain, Y. LeCun, and C. Bregler, "Joint training of a convolutional network and a graphical model for human pose estimation," *CoRR*, vol. abs/1406.2984, 2014.

[11] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915–1929, August 2013.

[12] T. N. Sainath, A. r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8614–8618.

[13] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, November 2012.

[14] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. H. Černocký, "Strategies for training large scale neural network language models," in *IEEE Automatic Speech Recognition and Understanding Workshop*, December 2011.

[15] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik, "Deep neural nets as a method for quantitative structureactivity relationships," *Journal of Chemical Information and Modeling*, vol. 55, no. 2, pp. 263–274, 2015, pMID: 25635324.

[16] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[18] Y. Bengio, A. C. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," *CoRR*, vol. abs/1206.5538, 2012.

[19] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50.

[20] B. S. Goh, W. Leong, and K. Teo, "Robustness of convergence proofs in numerical methods in unconstrained optimization," *Intelligent Systems, Control and Automation: Science and Engineering*, vol. 72, pp. 1–9, January 2014.

[21] B. S. Goh, "Approximate greatest descent methods for optimization with equality constraints," *Journal of Optimization Theory and Applications*, vol. 148, no. 3, pp. 505–527, 2011.

[22] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 343–351.

[23] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, July 2011.

[24] H. H. Tan, K. H. Lim, and H. G. Hendra, "Stochastic diagonal approximate greatest descent in neural network in neural networks," in *Proceedings of the IEEE International Joint Conference in Neural Networks*, 2017, pp. 1895–1898.

[25] N. S. Keskar and R. Socher, "Improving generalization performance by switching from adam to SGD," *CoRR*, vol. abs/1712.07628, 2017.

[26] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," *CoRR*, vol. abs/1902.09843, 2019.

[27] J. Nocedal and S. J. Wright, *Numerical Optimization, second edition*. World Scientific, 2006.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research,

Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[29] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends ® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[30] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147.

[31] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings in Computational Statistics (COMPSTAT)*, January 2010, pp. 177–186.

[32] G. E. Hinton, "Relaxation and its role in vision," Ph.D. dissertation, University of Edinburgh, 1978.

[33] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $o(1/\sqrt{k})$," *Soviet Mathematic Doklady*, vol. 27, pp. 372–376, 1983.

[34] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525 – 533, 1993.

[35] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[36] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, March 1989.

[37] J. Sohl-Dickstein, B. Poole, and S. Ganguli, "Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods," in *Proceedings of the 31st International Conference on International Conference on Machine Learning*. JMLR, 2014, pp. II–604–II–612.

[38] Y. Xiao, Z. Wei, and Z. Wang, "A limited memory bfgs-type method for large-scale unconstrained optimization," *Computers & Mathematics with Applications*, vol. 56, no. 4, pp. 1001 – 1009, 2008.

[39] C. C. A. Floudas and P. M. Pardalos, *Encyclopedia of Optimization*. Berlin, Heidelberg: Springer-Verlag, 2006.

[40] B. M. Wilamowski and H. Yu, "Improved computation for levenberg-marquardt training," *IEEE Transactions on Neural Networks*, vol. 21, no. 6, pp. 930–937, June 2010.

[41] J. Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 735–742.

[42] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[44] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, November 1998.

[45] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[46] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[47] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1058–1066.

[48] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *CoRR*, vol. abs/1505.00387, 2015.

[49] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, August 2003, pp. 958–963.

[50] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," *CoRR*, vol. abs/1605.07262, 2016.

[51] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. USA: Omnipress, 2011, pp. 265–272.

[52] R. Pascanu, Y. N. Dauphin, S. Ganguli, and Y. Bengio, "On the saddle point problem for non-convex optimization," *CoRR*, vol. abs/1405.4604, 2014.

[53] S. Amari, H. Park, and K. Fukumizu, "Adaptive method of realizing natural gradient learning for multilayer perceptrons," *Neural Computation*, vol. 12, no. 6, pp. 1399–1409, 2000.

[54] D. Xie, J. Xiong, and S. Pu, "All you need is beyond a good init: Exploring better solution for training extremely deep convolutional neural networks with orthonormality and modulation," *CoRR*, vol. abs/1703.01827, 2017.

[55] S. Zhai, Y. Cheng, W. Lu, and Z. Zhang, "Doubly convolutional neural networks," *CoRR*, vol. abs/1610.09716, 2016.

[56] S. Luan, B. Zhang, C. Chen, X. Cao, J. Han, and J. Liu, "Gabor convolutional networks," *CoRR*, vol. abs/1705.01450, 2017.

[57] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015.

[58] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," *CoRR*, vol. abs/1603.08029, 2016.

[59] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *CoRR*, vol. abs/1511.00363, 2015.

[60] J. M. Leon Yao, "Tiny imagenet classification with convolutional neural networks," Stanford University, Tech. Rep., 2015.

[61] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, 2016.

[62] B. S. Goh, "Greatest descent algorithms in unconstrained optimization," *Journal of Optimization Theory and Applications*, vol. 142, no. 2, pp. 275–289, 2009.

[63] B. S. Goh, "Numerical method in optimization as a multi-stage decision control system," *Latest Advances in Systems Science and Computational Intelligence*, pp. 25–30, 2012.

[64] C. N. L. Eu, "Numerical analysis in nonlinear least squares methods and applications," Master's thesis, Curtin University, Department of Electrical and Computer Engineering, 2017.

[65] M. S. Lee, "Control system approach for constructing numerical methods in optimization and applications," Ph.D. dissertation, Curtin University, Department of Electrical and Computer Engineering, 2018.

[66] M. S. Lee, B. S. Goh, H. G. Harno, and K. H. Lim, "On a two-phase approximate greatest descent method for nonlinear optimization with equality constraints," *Numerical Algebra, Control & Optimization*, vol. 8, pp. 325–336, January 2018.

[67] S. Becker and Y. Lecun, "Improving the convergence of back-propagation learning with second-order methods," in *Proceedings of the 1988 Connectionist Models Summer School, San Mateo*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. Morgan Kaufmann, 1989, pp. 29–37.

[68] Y. Lecun, *Generalization and network design strategies*. Elsevier, 1989.

[69] C. Bishop, "Exact calculation of the Hessian matrix for the multilayer perceptron," *Neural Computation*, vol. 4, no. 4, pp. 494–501, July 1992.

[70] A. M. Lyapunov, "The general problem of the stability of motion," *International Journal of Control*, vol. 55, no. 3, pp. 531–534, 1992.

[71] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems 19*, P. B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 153–160.

[72] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, December 2010.

[73] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, July 2006.

[74] H. Poon and P. M. Domingos, "Sum-product networks: A new deep architecture," *CoRR*, vol. abs/1202.3732, 2012.

[75] O. Delalleau and Y. Bengio, "Shallow vs. deep sum-product networks," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 666–674.

[76] 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, December 2010.

[77] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th International Conference on Computer Vision*, September 2009, pp. 2146–2153.

[78] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, May 2010, pp. 253–256.

[79] L. Deng, "An overview of deep-structured learning for information processing," in *Proceedings of the Asian-Pacific Signal & Information Processing Annual Summit & Conference (APSIPA-ASC)*, October 2011, pp. 1–14.

[80] L. Deng, D. Yu, and J. Platt, "Scalable stacking and learning for building deep architectures," in *Proceedings of International Conference on Acoustics Speech and Signal Processing (ICASSP)*, March 2012.

[81] G. Tur, L. Deng, D. Hakkani-Tür, and X. D. He, "Towards deeper understanding deep convex networks for semantic utterance classification," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, March 2012.

[82] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 436, p. 436444, May 2015.

[83] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[84] 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.

[85] M. M. Lau and K. H. Lim, "Review of adaptive activation function in deep neural network," in *2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, Dec 2018, pp. 686–690.

[86] C. Cai, Y. Xu, D. Ke, and K. Su, "Deep neural networks with multistate activation functions," *Computational intelligence and neuroscience*, vol. 2015, p. 721367, September 2015.

[87] G. Yang and S. S. Schoenholz, "Mean field residual networks: On the edge of chaos," *CoRR*, vol. abs/1712.08969, 2017.

[88] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the 30 th International Conference on Machine Learning*, 2013.

[89] Jaewan-Yun, "Optimizer visualization," https://github.com/Jaewan-Yun/optimizer-visualization/blob/master/LICENSE, accessed: 2019-07-31.

[90] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[91] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.