

OPTIMISING NEURAL NETWORK TRAINING  
EFFICIENCY THROUGH SPECTRAL  
PARAMETER-BASED MULTIPLE ADAPTIVE  
LEARNING RATES

KOAY YEONG LIN

MASTER OF SCIENCE

LEE KONG CHIAN FACULTY OF ENGINEERING AND  
SCIENCE  
UNIVERSITI TUNKU ABDUL RAHMAN  
OCTOBER 2023

**OPTIMISING NEURAL NETWORK TRAINING EFFICIENCY  
THROUGH SPECTRAL PARAMETER-BASED MULTIPLE  
ADAPTIVE LEARNING RATES**

By

**KOAY YEONG LIN**

A dissertation submitted to the  
Department of Mathematical and Actuarial Sciences,  
Lee Kong Chian Faculty of Engineering and Science,  
Universiti Tunku Abdul Rahman,  
in partial fulfillment of the requirements for the degree of  
Master of Science  
October 2023

## ABSTRACT

### OPTIMISING NEURAL NETWORK TRAINING EFFICIENCY THROUGH SPECTRAL PARAMETER-BASED MULTIPLE ADAPTIVE LEARNING RATES

Koay Yeong Lin

The process of training deep neural networks involves heavily solving optimization problems. Finding optimal values for different hyperparameters makes training neural networks challenging. A hyperparameter called learning rate or step size is one of the most crucial factors in optimization using gradient-based approaches. A small learning rate might result in slow convergence and the loss function will get stuck in the local minimum, whereas a large learning rate might hinder convergence or cause divergence. Currently, most of the common optimization algorithms use a fixed learning rate or a simplified adaptive updating scheme in every iteration. In this project, we propose a stochastic gradient descent method with multiple adaptive learning rates (MAdaGrad) and Adam with multiple adaptive learning rates (MAdaGrad Adam). In the derivation of the updating formula, we aim to minimize the log-determinant norm and allow them to satisfy the secant equation. We apply the Lagrange multiplier to the minimization problem and the Lagrange multiplier can be approximated by using the Newton-Raphson method. The proposed algorithms update the learning rate in every iteration based on the approximated spectrum of the Hessian of the loss function. The methods were compared to the existing optimization methods in deep learning, stochastic gradient descent method (SGD) and Adam. Some datasets were used to observe the performance of the proposed

methods. The numerical results show that the proposed methods perform better than SGD and Adam. Hence, the proposed MAdaGrad and MAdaGrad Adam can be alternative optimizer in machine learning.

## ACKNOWLEDGMENTS

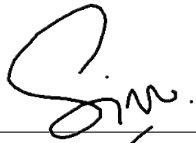
I would like to thank the following people, without whom I would not have been able to complete this research, and without whom I would not have made it through my study. My most profound appreciation goes to Dr. Sim Hong Seng, Dr. Goh Yong Kheng and Dr. Chua Sing Yee, my supervisor and co-supervisors, for their time, effort, and understanding in helping me succeed in my studies. Their vast wisdom and wealth of experience have inspired me throughout my studies.

Furthermore, I value the love and encouragement of my extended family for providing me with unfailing support and continuous encouragement throughout my study. Finally, I'd like to express my gratitude to UTAR for providing me with the scholarship that allowed me to complete this project.

## APPROVAL SHEET

This dissertation entitled "OPTIMISING NEURAL NETWORK TRAINING EFFICIENCY THROUGH SPECTRAL PARAMETER-BASED MULTIPLE ADAPTIVE LEARNING RATES" was prepared by KOAY YEONG LIN and submitted as partial fulfillment of the requirements for the degree of Master of Science at Universiti Tunku Abdul Rahman.

Approved by:



\_\_\_\_\_  
(Dr Sim Hong Seng)  
Assistant Professor/Supervisor  
Department of Mathematical and Actuarial Sciences  
Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman

Date: 10 October 2023



\_\_\_\_\_  
(Dr Goh Yong Kheng)  
Associate Professor/Co-supervisor  
Department of Mathematical and Actuarial Sciences  
Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman

Date: 10 October 2023



\_\_\_\_\_  
(Dr Chua Sing Yee)  
Assistant Professor/Co-supervisor  
Department of Electrical and Electronic Engineering

Date: 10 October 2023

Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman

LEE KONG CHIAN FACULTY OF ENGINEERING AND SCIENCE

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 10 October 2023

**SUBMISSION OF DISSERTATION**

It is hereby certified that **Koay Yeong Lin** (ID No: **21UEM01866** ) has completed this dissertation entitled “OPTIMISING NEURAL NETWORK TRAINING EFFICIENCY THROUGH SPECTRAL PARAMETER-BASED MULTIPLE ADAPTIVE LEARNING RATES” under the supervision of Dr Sim Hong Seng (Supervisor) from the Department of Mathematical and Actuarial Sciences, Lee Kong Chian Faculty of Engineering and Science and Dr Goh Yong Kheng (Co-supervisor) from the Department of Mathematical and Actuarial Sciences, Lee Kong Chian Faculty of Engineering and Science.

I understand that the University will upload softcopy of my dissertation in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,



---

Koay Yeong Lin



## DECLARATION

I **KOAY YEONG LIN** hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.



---

(KOAY YEONG LIN)

Date: 10 October 2023

## TABLE OF CONTENTS

	<b>Page</b>
<b>ABSTRACT</b>	<b>ii</b>
<b>ACKNOWLEDGMENTS</b>	<b>iv</b>
<b>APPROVAL SHEET</b>	<b>v</b>
<b>SUBMISSION SHEET</b>	<b>vii</b>
<b>DECLARATION</b>	<b>viii</b>
<b>LIST OF TABLES</b>	<b>xi</b>
<b>LIST OF FIGURES</b>	<b>xii</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xiv</b>
<b>CHAPTER</b>	
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Introduction	1
1.2 Objective	2
1.3 Problem Statement	3
1.4 Structure of the Thesis	4
<b>2 LITERATURE REVIEW</b>	<b>5</b>
2.1 Mathematical Optimization	5
2.2 Deep Learning	10
2.3 Gradient-Based Optimization Techniques in Deep Learning	11
2.4 Learning Rate Schedules in Deep Learning	18
2.5 Summary	21
<b>3 METHODOLOGY</b>	<b>22</b>
3.1 Mathematical Formulation of Multiple Adaptive Learning Rate Algorithms	22
3.1.1 Multiple Adaptive Learning Rate Incorporated to SGD	25
3.1.2 Multiple Adaptive Learning Rate Incorporated to Adam	25
3.1.3 Convergence Analysis	26
3.2 Implementation	28
3.2.1 Implementation of sklearn	28
3.2.2 SGD	30
3.2.3 MAdaGrad	31

3.2.4	<b>MAdaGrad Adam</b>	34
3.3	<b>Benchmark</b>	35
<b>4</b>	<b>NUMERICAL RESULTS AND DISCUSSIONS</b>	<b>37</b>
4.1	<b>Datasets</b>	37
4.2	<b>Experimental Setup</b>	38
4.3	<b>Comparison between MAdaGrad and SGD - Loss values</b>	40
4.3.1	<b>MNIST handwritten digits datasets</b>	40
4.3.2	<b>Abalones datasets</b>	44
4.3.3	<b>Breast cancer datasets</b>	44
4.3.4	<b>Wine datasets</b>	46
4.3.5	<b>Combined Datasets</b>	50
4.4	<b>Comparison between MAdaGrad Adam and Adam - Loss values</b>	53
4.4.1	<b>MNIST handwritten digits datasets</b>	53
4.4.2	<b>Abalone datasets</b>	53
4.4.3	<b>Breast cancer datasets</b>	53
4.4.4	<b>Wine datasets</b>	60
4.4.5	<b>Combined Datasets</b>	60
4.5	<b>Comparison between MAdaGrad and SGD - Number of Iterations</b>	64
4.6	<b>Comparison between MAdaGrad Adam and Adam - Number of Iterations</b>	64
4.6.1	<b>Loss values for MAdaGrad and SGD(constant)</b>	64
4.6.2	<b>Loss values for MAdaGrad Adam and Adam</b>	67
4.7	<b>Restoring, Restarting and Rescaling</b>	67
<b>5</b>	<b>CONCLUSION AND FUTURE WORKS</b>	<b>70</b>
5.1	<b>Conclusion</b>	70
5.2	<b>Future Work</b>	71
	<b>Bibliography</b>	<b>73</b>

## LIST OF TABLES

<b>Table</b>	<b>Page</b>
4.1 Attribute Information of Breast Cancer Datasets	38
4.2 Attribute Information of Wine Datasets	39
4.3 Attribute Information of Abalone Datasets	39

## LIST OF FIGURES

<b>Figure</b>	<b>Page</b>
4.1 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (MNIST Dataset).	42
4.2 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (MNIST Dataset).	43
4.3 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Abalone Dataset).	44
4.4 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Abalone Dataset).	45
4.5 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Breast Dataset).	46
4.6 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Breast Dataset).	47
4.7 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Wine Dataset).	48
4.8 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Wine Dataset).	49
4.9 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Combined Dataset).	51
4.10 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Combined Dataset).	52
4.11 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (MNIST Dataset).	54
4.12 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (MNIST Dataset).	55

4.13 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Abalone Dataset).	56
4.14 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Abalone Dataset).	57
4.15 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Breast Dataset).	58
4.16 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Breast Dataset).	59
4.17 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Wine Dataset).	60
4.18 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Wine Dataset).	61
4.19 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Combined Dataset).	62
4.20 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Combined Dataset).	63
4.21 Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (MNIST Dataset - Number of Iterations).	65
4.22 Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (MNIST Dataset - Number of Iterations).	66
4.23 Loss over iterations with hidden layer size (100,). (Top) MAdaGrad and SGD(constant). (Bottom) MAdaGrad Adam and Adam	68
4.24 Restoring, Restarting and Rescaling	68
4.25 Count in Restoring and Restarting.	69

## LIST OF ABBREVIATIONS

<b>SGD</b>	Stochastic Gradient Descent method
<b>Adam</b>	Adaptive Moment Estimation method
<b>MAdaGrad</b>	Stochastic Gradient Descent method with multiple adaptive learning rates
<b>MAdaGrad Adam</b>	Adam method with multiple adaptive learning rates
<b>NAG</b>	Nesterov Accelerated Gradient method
<b>AdaGrad</b>	Adaptive Gradient method
<b>RMSProp</b>	Root Mean Squared Propagation

## CHAPTER 1

### INTRODUCTION

#### 1.1 Introduction

Neural networks can be improved in a variety of ways, including by enhancing the architecture, experimenting with data representation, finding the optimal parameters, and selecting the best optimization technique. There are currently no standards for constructing an optimal machine learning architecture (Tato and Nkambou, 2018).

Training neural models involve optimizing the scalar-parameterized objective function, aiming to minimize the loss function based on the model parameters, thus, it can be considered a challenging optimization problem (Krizhevsky et al., 2012). The key hyperparameter to adjust when training neural networks is the learning rate, as is well-established. A large learning rate could hinder convergence and diverge, and the loss function might fluctuate in contrast to a smaller learning rate could lead to slow convergence, and the loss function might get stuck at a local minimum.

A wide variety of engineering and science areas rely heavily on stochastic gradient-based optimization. In these areas, numerous problems can be treated as the maximization or minimization of an objective function with scalar parameters and concerning its parameters (Kingma and Ba, 2014). The minimization involves determining the set of parameters that yield the greatest results in tasks like classification,



regression, and clustering. First-order optimization techniques have been utilized for training many machine learning models with good performance and efficient computation, including neural networks (Bae et al., 2019).

The minimization problem in the weights of a neural network occurs in many machine learning problems. Consider the problem of determining a neural network that learns the relationship between inputs  $x_i$  and outputs  $y_i$  using training data  $(x_1, y_1), \dots, (x_n, y_n)$ . This relates to a function  $y = F(\theta, x)$  that minimizes the average loss function and is parameterized by the parameters over the training data:

$$f(\theta) = \frac{1}{N} \sum_{i=1}^N l(x_i, y_i, \theta) = \frac{1}{N} \sum_{i=1}^N f_i(\theta) \quad (1.1)$$

## 1.2 Objective

The project aims to propose stochastic gradient descent with multiple adaptive learning rates for neural network. The proposed algorithm will be compared with the standard conventional method, by comparing their efficiency and performance in terms of the number of iterations and loss values. The standard conventional algorithm applies a fixed learning rate or a simplified adaptive updating scheme in every iteration.

The project's specific goals include:

1. Deriving an updating formula for multiple learning rates to be incorporated with the stochastic gradient descent method for machine learning.
2. Establish the convergence properties of the proposed algorithm.
3. Develop a program to validate the efficiency of the proposed algorithm.

### 1.3 Problem Statement

The process of training neural networks involves heavily in solving optimization problems. Finding optimal values for different hyperparameters makes training neural networks challenging. When implementing gradient-based techniques, one of the most important hyperparameters is the learning rate, also known as step size.

Due to its simplicity, stochastic gradient descent (SGD) is currently one of the most common methods and is relatively efficient. SGD has the limitation of scaling the gradient uniformly in all directions due to the fixed learning rate utilized throughout the simulation. As a result, the initial learning rate requires to be manually adjusted. The convergence speed will be reduced, and the training algorithm may even diverge if an inappropriate learning rate is chosen. Changes in the learning rate will also affect the model performance or the classification accuracy. As a result, the main difficulties in the training process for SGD include fine-tuning the learning rate and figuring out the optimum learning rate.

A proper selection of the learning rate determines how well SGD performs, and it is currently the most popular research topic. To improve learning performance, adaptive optimization techniques like AdaGrad, Adam, and RMSProp have been developed. These techniques may adjust the learning rate during the optimization process. Although different adjusted learning rates are utilized for training in each iteration of these adaptive learning rate approaches, the same learning rate is used for every weight component and bias component.

Currently, the common existing learning rate algorithms are either using a fixed learning rate in the whole simulation or adapting the learning rate in every iteration. We suggest developing a stochastic gradient descent with multiple adaptive learning rates for machine learning in this project.

The proposed algorithm will adapt the learning rate in every iteration as well as in every single equation, i.e. every weight component and bias component. The proposed method will be compared to the existing optimization methods in machine learning, such as Adam and SGD.

#### **1.4 Structure of the Thesis**

An introduction to optimization and neural networks are discussed in the first chapter. This chapter also includes the project's objective and problem statements.

The literature is reviewed in Chapter 2. The reviews have covered some reviews on mathematical optimization, gradient-based optimization techniques and some learning rate schedules.

Chapter 3 involves the derivation of multiple adaptive learning rate algorithms and its convergence analysis. Multiple adaptive learning rates incorporated into SGD and Adam will also be discussed in this chapter.

Chapter 4 shows numerical results and discussion by making comparisons between the proposed methods and the existing methods while the conclusion of the entire work and the possible future work will be involved in Chapter 5.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Mathematical Optimization

In both optimization theory and algorithm design, large-scale optimization is a key research area. For instance, efficient approaches for solving optimization problems include the conjugate gradient method, spectral gradient method, and quasi-Newton method. Consider the subsequent unconstrained optimization problem:

$$\min f(x), x \in R^n \tag{2.1}$$

where  $f : R^n \rightarrow R$  is a continuously differentiable function.

To guarantee the consistent improvement of the objective function  $f$ , these algorithms are typically combined with the line search method (Le et al., 2011). The model is gradually enhanced using an optimization method until it can no longer be improved or until the estimated cost or time has been met.

Cauchy et al. (1847) first introduced the steepest descent method. It is straightforward to implement, has little storage, and requires low computational resources. The steepest descent method, however, is known to yield poor performance. The method's oscillatory characteristic and slow convergence rate are the main reason for its inefficiency. It is significantly affected by ill-conditioning problems. The

theory and construction of optimization techniques still rely on an understanding of the characteristics of the steepest descent method.

Conjugate gradient algorithms perform well in addressing optimization problems. Because of their simplicity, speedy convergence, and minimal memory requirements, they are particularly well-liked. The computation and storage of several matrices related to the Hessian of objective functions can be avoided using this method, similar to the steepest descent technique. A strategy for dealing with positive definite and symmetric linear systems is the conjugate gradient method (Hestenes and Stiefel, 1952).

The Newton approach is one of the best ways to solve unconstrained optimization problems. It usually needs only a few function evaluations and is effective in dealing with ill-conditioning problems. (Xiao et al., 2008). Nevertheless, the method's effectiveness depends heavily on the method's ability to solve a linear problem effectively, which happens while determining the search direction for each iteration (Xiao et al., 2008).

For solving minimization problems, numerous techniques are the modifications of the Newton approach, that necessitate defining the second derivative of an objective function, the Hessian matrix. Unconstrained optimization problems can be addressed effectively with quasi-Newton methods since they have been demonstrated to be robust and has inexpensive computation. The application in plenty of numerical optimization fields has been heavily encouraged by their superior characteristics (Morales, 2002). This method works by removing the previous information and replacing it with the latest information (Nocedal, 1980). When the computation of the Hessian matrix is challenging or expensive, quasi-Newton methods will be an alternative algorithm.

The shortcoming of the quasi-Newton method is that in every iteration, it requires computing and storing a full rank matrix. A series of matrices are generated to approximate the hessian or its inverse. These matrices require  $\frac{n(n+1)}{2}$  storage because they are symmetric matrices (Nocedal, 1980), where  $n$  is the length of the corresponding matrix. Therefore, it requires high computational costs when dealing with large-scale problems.

In terms of computation, the standard Broyden-Fletcher-Goldfarb-Shanno (BFGS) approach has been demonstrated to be the most successful quasi-Newton method. The BFGS approach has been shown to be trustworthy and robust. The outstanding convergence characteristics of Newton's method are preserved by the BFGS method. Only the gradient information is utilized by the BFGS update, whereas the available function values are ignored (Xiao et al., 2008). However, Quasi-Newton techniques can be utilized for dealing with medium- and small-scale optimization problems where the approximate inverse Hessian matrix exists.

Nocedal (1980) presented a limited memory BFGS approach (L-BFGS) for unconstrained optimization problems to address the weakness of BFGS. A modification of the BFGS approach for solving large-scale problems is known as L-BFGS. It was suggested as a method to avoid storing the matrix instead of reconstructing them by using knowledge of the most  $m$  recent steps (Nocedal, 1980). The difference between the BFGS approach and the L-BFGS is the implementation part. In L-BFGS, the inverse Hessian approximation is formed by using a few BFGS updates rather than being formed explicitly. L-BFGS preserves a compact approximation of the Hessian with a low storage requirement (Xiao et al., 2008).

The L-BFGS approach is similar to the BFGS approach for the first  $m$

iterations, but it retains BFGS corrections independently until the maximum  $m$  reaches. The most recent corrections are then added in replacement of the previous ones. Performing  $m$  BFGS updates on the user-provided sparse symmetric and positive definite matrix  $H_0$  gives the approximation of the inverse Hessian of  $f$  at iteration  $k$ ,  $H_k$ . Due to its minimal storage requirements and rapid rate of linear convergence, the L-BFGS approach is indeed very efficient, according to numerical studies (Liu and Nocedal, 1989). L-BFGS has drawn a lot of attention in recent years (Liu et al., 2022);(Berahas et al., 2022).

Andrei (2018) has introduced a quasi-Newton approach using the diagonal updates matrices. The measure function of Byrd and Nocedal is minimized to produce the diagonal matrix's component parts. A novel diagonal quasi-Newton updating algorithm based on the prior technique has been proposed by Andrei (2019). The elements of the diagonal matrix are obtained from the gradient component's scaled directional derivatives forward finite difference. The updating method for accelerated diagonal quasi-Newton is also established for unconstrained optimization. The directional derivatives for the forward finite difference of the gradient elements are scaled to produce the diagonal matrix elements that approximate the Hessian (Andrei, 2021).

Besides, Barzilai-Borwein is the origin of spectral gradient (SG) approaches for minimization. A technique known as the two-point step size gradient approach was proposed by Barzilai and Borwein (1988). The spectral gradient method's quadratic convergence has been created by Raydan (1993). The spectral gradient approach is one of the most prominent techniques for large-scale problems. To address the shortcomings of the Cauchy method, this strategy uses a nonmonotone step length connected to the gradient technique. Since there are numerous options for selecting an appropriate step size in the negative

gradient direction, many different strategies have been suggested (Biglari and Solimanpur, 2013). This approach is created by approximating the secant equation for the SD method. The objective function for the SG technique doesn't guarantee a descent at each iteration, but in practice, it works better than the standard SD approach (Raydan, 1997). Given that it only requires a few storage locations, this method outperforms the standard SD method in terms of computation efficiency and cost. The efficiency of the technique can be significantly improved by combining the traditional SG approach with superior nonmonotone line search techniques (Xiao et al., 2010).

The gradient approaches will be ineffective if the objective function's Hessian matrix is poorly constrained. The gradient methods have a set requirement in the choice of step length, which reduces the function value. Sim et al. (2019) has improved the SG approach in order to address the issue of inefficiency. The slow convergence concerns are resolved using this technique. Both the objective function and the norm of the gradient vector are simultaneously managed independently. Additionally, various line search techniques are coupled with this strategy. While a single adaptive parameter dampens the gradient vector, and the line search is employed to reduce the evaluation of the function. The proposed strategy is generated using nonmonotone line search and backtracking line search. Given that CG-based methods have outstanding convergence characteristics, a comparison is conducted between the proposed method and a few well-known CG-based techniques. The proposed spectral gradient technique is demonstrated to be a competitive alternative for handling large-scale problems by Sim et al. (2019).



## 2.2 Deep Learning

A subset of machine learning called "deep learning" uses hierarchical designs to extract high-level abstractions from data. It is a novel technique that has been widely applied in well-established artificial intelligence domains, such as computer vision (Bhargava and Bansal, 2021), natural language processing (Chen et al., 2021), healthcare (Lee and Yoon, 2021) and many others. Deep learning consists of multiple layers to separate higher-level features from the input's raw data. For instance, an image is composed of a set of pixel values and is processed through the deep neural network. In the first layer representation, the learned features indicate the number of edges present in the image. Considering slight changes in the edge placements, the second layer finds patterns by identifying specific combinations of edges. The third layer combines patterns into bigger combinations that relate to parts of some well-known objects. The following layers identify items as combinations of the pieces in the third layer. Instead of being generated by human professionals, these layers of attributes are learned through data using a general-purpose learning strategy. (LeCun et al., 2015)

Deep learning is currently growing for three primary reasons: the tremendous developments in machine learning techniques, the significantly enhanced abilities of chip processing, and the substantially reduced cost of computing hardware (Guo et al., 2016). Concerning problems that have long resisted the best efforts of the artificial intelligence sector, deep learning is making tremendous progress. It can be applied in a variety of scientific, commercial, and governmental domains since it is efficient at finding complicated structures in high-dimensional data. Many deep learning algorithms have been extensively examined and investigated in recent years.

## 2.3 Gradient-Based Optimization Techniques in Deep Learning

Gradient descent is one of the most traditional techniques introduced by Cauchy. It was significant in the area of optimization. Gradient descent starts with an initial vector  $\theta_0$  and updates it as follows:

$$\theta_{k+1} = \theta_k - \alpha_k \cdot \nabla f(\theta_k), \quad \text{for } k = 0, 1, \dots, N - 1 \quad (2.2)$$

where  $\nabla f(\theta_k)$  is the gradient of loss function at  $\theta_k$  and  $\alpha_k$  is the learning rate.

The process of learning in deep learning can be optimized by using some optimization approaches based on the gradient descent method. Gradient descent for deterministic optimization (Cartis et al., 2010);(Nesterov, 2003) and stochastic gradient descent for stochastic optimization (Ghadimi and Lan, 2013);(Zinkevich, 2003) are the two most well-known first-order techniques. Gradient descent determines  $\Delta t$  using the full batch gradient of the objective function and stochastic gradient descent uses a stochastic gradient estimate, which is simpler but more computationally efficient.

Most practical deep neural network optimization approaches are based on the stochastic gradient descent (SGD) method (Zhang, 2018). The simplicity of stochastic gradient descent (SGD) (Robbins and Monro, 1951) makes it effective in a wide range of application sectors. It is one of the most powerful first-order optimization methods. SGD uses linear iteration to approximate optimal solutions, which is a straightforward process for each iteration.

SGD updates its parameters for every training example  $x_i$  and label  $y_i$  :

$$\theta_{k+1} = \theta_k - \alpha \cdot \nabla_{\theta} J(\theta_k; x_i; y_i). \quad (2.3)$$

where  $\nabla_{\theta}J$  represent the gradient of the loss function.

When dealing with very large datasets, the computational complexity of learning methods becomes an important limiting factor. SGD can be simply applied even to problems with complex structures and high-dimensional parameters. As a result, SGD has become a widely used optimization approach for large-scale optimization problems.(Zhang et al., 2020)

However, SGD has a main weakness which is the slow convergence rate. Many researchers have aimed to improve SGD by accelerating the convergence rate in different ways, such as improving the parameters of the update method or making adjustments to the learning rate. Roux et al. (2012) introduced a novel stochastic gradient approach for the optimization of the strongly convex sum of a finite set of smooth functions. To determine a linear convergence rate, the new method leverages a recollection of historical gradient values. They have performed some numerical tests compared with some existing methods, such as the stochastic gradient method, stochastic average gradient and the full gradient method. The results revealed that the new approach can significantly outperform traditional machine learning methods, both concerning minimizing the test error efficiently and optimizing the training loss.

A method for explicitly reducing variance in stochastic gradient descent algorithms referred to as stochastic variance reduced gradient (SVRG), was suggested by Johnson and Zhang (2013). The reason of the proposing is that stochastic gradient descent has a slow convergence rate as a result of the inherent variance. They demonstrated this method has a rapid convergence rate, similar to the methods that improve SGD by reducing the variance, such as stochastic average gradient (SAG) and stochastic dual coordinate ascent (SDCA). Since the novel approach

does not rely on gradient storage, it can be used to solve a variety of challenging issues, including some structured prediction problems and training neural.

However, backpropagation using SGD requires manual adjustment of the initial learning rate. Since the magnitudes of various parameters vary widely and adjustment is needed throughout the training process, the learning rate of SGD can be challenging to tune. The convergence speed will be reduced by an inappropriate learning rate and result in the divergence of the training method (Montavon et al., 2012). Variations in the learning rate have a direct impact on how well the model performs. The main challenges in SGD training revolve around optimizing and modifying the learning rate. Numerous accelerated SGD variations have so been introduced in the recent past as a result.

Recent studies have presented a number of accelerated variations of gradient descent and stochastic gradient descent. Such variants can be divided into three types, which include momentum methods that design the descent direction carefully, adaptive learning rate methods that decide good learning rates, and adaptive gradient methods that combine the strengths of the first and second types of variants. (Chen et al., 2018)

The momentum method (Liu et al., 2020);(Ghadimi et al., 2015); (Nesterov, 1983);(Polyak, 1964) has accelerated the descent in pertinent directions while slowing it down in irrelevant ones.

$$v_k = \gamma v_{k-1} - \alpha \cdot \nabla_{\theta} J(\theta) \quad (2.4)$$

where  $\gamma v_{k-1}$  is the momentum term. However, SGD with momentum tends to get stuck on data with unequally distributed data points. Therefore, Nesterov Accelerated Gradient (NAG) is proposed to overcome the weakness of the method SGD with momentum. NAG is a

hyper-parameter that indicates where the global optimal value is located. The formula of NAG is shown below:

$$v_k = \gamma v_{k-1} + \alpha \cdot \nabla_{\theta} J(\theta - \gamma \cdot v_{k-1}) \quad (2.5)$$

$$\theta_{k+1} = \theta_k - v_k \quad (2.6)$$

where  $\theta - \gamma \cdot v_{k-1}$  is determined, and provides an estimate of the parameters' next approximate position. Momentum determines the current gradient before leaping the updated accumulated gradient's direction. Contrarily, NAG performs jumps in the previously accumulated gradient's direction, before measuring and adjusting the gradient. This type of predictive update prevents faster convergence in the wrong direction while also improving the performance (Lydia and Francis, 2019).

Adaptive methods have been presented as SGD variations that can use different learning rates for each parameter. Duchi et al. (2011) introduced the Adaptive Gradient Descent Algorithm (AdaGrad). By dividing the learning rate for each parameter by the square root of the gradient vector's sum of squares, AdaGrad effectively adapts each parameter's learning rate. AdaGrad maintains small learning rates for frequently occurring features while large learning rates for less frequently occurring features, making it suited for sparse data ((Dean et al., 2012) and (Pennington et al., 2014)).

At each step  $t$ , AdaGrad modifies the general fixed learning rate for every parameter  $\theta_k$  based on the previous gradients computed for  $\theta_k$ . The updating formula of AdaGrad is

$$\theta_{k+1} = \theta_k - \frac{\alpha}{\sqrt{G_k + \epsilon}} \cdot \nabla_{\theta} J(\theta_k; x_i, y_i) \quad (2.7)$$

where  $\nabla_{\theta} J(\theta_k; x_i, y_i) = g_k$  is the loss function gradient with respect to the parameter  $\theta_k$  at time step  $t$  and  $\epsilon$  is a value added to avoid division by zero. Each diagonal element  $i$  in the diagonal matrix  $G_k$  is equal to the sum of the squares of the gradients with respect to  $\theta_k$ . However, this method has its main weakness. Since the squared gradients are all positive values, as the time step  $t$  increases, the learning rate decreases until it becomes infinitely small as the squared gradients in the denominator accumulate. (Ruder, 2016)

RMSprop is proposed by Tieleman et al. (2012). This well-known method is an unpublished gradient-based optimization method used in training neural networks. RMSProp improves AdaGrad to prevent the problem of monotonically decreasing the learning rate. RMSprop fixed the problem by using an average scale rather than a cumulative scale (Bock and Weiß, 2019). The gradient is normalized by dividing the learning rate by an exponentially decaying average of squared gradients (Kochenderfer and Wheeler, 2019). It increases the small gradient's learning rate to prevent vanishing and decreases the large gradient's learning rate to prevent exploding.

AdaDelta (Zeiler, 2012), is a powerful methodology for learning rates, and it can be implemented in a range of circumstances. The concept originates from AdaGrad (Duchi et al., 2011) and the method is introduced to deal with the issues of learning rates that continuously decay and the selection of hyperparameters. Instead of summing up all the preceding gradients, AdaDelta (Zeiler, 2012) adjusts the learning rates depending on a moving window of gradient updates. The learning rate does not need to be tuned manually and without the need to assign the initial learning rate like AdaGrad (Duchi et al., 2011), since it was removed from the update rule.

Adam (Adaptive Moment Estimation) was first introduced by

Kingma and Ba (2014). It is simple to implement and does not need a large amount of memory. It is a stochastic optimization approach that uses only first-order gradients and is computationally efficient. The concept of momentum is used in this method, which multiplies the gradient values by the weights computed previously (Yi et al., 2020). This method combines the strength of two famous optimization approaches, which are AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman et al., 2012). Thus, Adam is well suited to non-stationary objectives and problems with sparse gradients. Adam has the advantage of performing a form of step size annealing naturally. Adam generates unique adaptive learning rates for distinct parameters using estimates of the first and second moments of the gradients. Due to its adaptive learning rate and good performance, Adam has been widely used in training deep neural networks. In the field of machine learning, A large variety of non-convex optimization problems are well-suited for Adam.

Adam uses an iterative learning approach to update the variables. The learning rate is adaptive due to the use of first-order momentum, and it uses second-order momentum to preserve a portion of the past gradient direction. The first-order estimate is described as

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (2.8)$$

where  $m_t$  is the gradient's exponential moving average,  $\beta_1$  is the exponential decay rate for the first moment estimate and  $g_t$  is the loss function gradient. The estimate of second-order moment is defined as

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (2.9)$$

where  $v_t$  is the exponential moving average of the gradient squared and  $\beta_2$  is the exponential decay rate for the second moment estimate.

Initially, the first-order and second-order moment estimates are set to be 0, and the  $\beta_1$  and  $\beta_2$  are close to 1, therefore the first-order estimate  $m_t$  and second-order estimate  $v_t$  are biased towards zero.

The bias correction of the first-moment and second-moment estimates are shown

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (2.10)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (2.11)$$

The updating formula is thus given as:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \quad (2.12)$$

where  $\alpha$  is the learning rate and  $\epsilon$  is set to be  $10^{-8}$ , which is recommended from the scikit-learn package.

However, some studies have highlighted the limitations of the Adam method (Reddi et al., 2019);(Luo et al., 2019);(Zhou et al., 2018). They demonstrated that Adam cannot converge to the optimal point in counterexamples and proposed revised adaptive approaches. Unfortunately, Reddi et al. (2019) discovered that several of Adam's lemmas could not be proven. Furthermore, Kingma and Ba addressed the problems raised and proposed a method called AMSGrad, a variant of Adam. The max operator is used for second momentum estimations in AMSGrad (Reddi et al., 2019).

Tran et al. (2019) demonstrated that AMSGrad's convergence proof also has a shortcoming and that Adam's convergence proof has a similar problem that has been overlooked. The handling of the hyper-parameters, which assumes them to be equal when they are not,



is the specific issue with AMSGrad’s convergence proof. To illustrate the overlooked problem, they gave an explicit counterexample of a simple convex optimization scenario. Several solutions to this problem relying on the manipulation of hyper-parameters have been provided. They presented a new convergence proof for AMSGrad and a modified version of AMSGrad, named AdamX.

AdaGrad, RMSProp, and Adam are some common adaptive optimization techniques that have been suggested to achieve a quick training process. Despite their popularity, they are shown to have a poorer generalization than SGD. Due to the extreme and unstable learning rate, these adaptive optimization methods may result in poor performance or even fail to converge. Therefore, new variations of AMSGrad and Adam, known as AMSBound and AdaBound, respectively, are presented by Luo et al. (2019). They utilized dynamic restrictions on the learning rates to establish a smooth and gradual switch from adaptive techniques to SGD. According to experimental outcomes, the new methods can maintain better learning speed early in training while also reducing the generalization difference between SGD and adaptive strategies. Additionally, they can perform much better on their prototypes, particularly for complex deep neural networks. The new methods demonstrated great performances on some common benchmarks while preserving beneficial aspects of adaptive methods like speedy initial progress and insensitivity of the hyperparameter.

## **2.4 Learning Rate Schedules in Deep Learning**

The learning rate has been scheduled using various approaches to address this problem, including time-based (Park et al., 2020), step-based (Ge et al., 2019) and exponential-based (Li and Arora, 2019) learning rate schedule. Generally, these methods involve additional

hyperparameters to manage speed for the decay of the learning rate. To prove the SGD convergence, reducing the step size is notable as a crucial element and assists in practical implementations of SGD. Throughout the training process, a successful schedule can adjust the learning rate over time. This will lead to better convergence and converges to a satisfactory minimum.

The existing learning rate schedules used are just reducing the learning rate, which might be a reason that resulted in the learning not proceeding further at a local minimum ((Becker and Le Cun, 1988); (Kelley, 1995)). When using these strategies, the outcome is frequently unpredictable and won't be close to zero for the cost function created from learning data. Therefore, by utilizing cost functions, Park et al. (2020) proposed a new schedule for learning rates. The cost function value is used to decide the adjustment of the learning rate, the approach can only stop learning at the cost function's global minimum, which indicates that the given cost reaches zero.

The learning rate range test (LR range test) and cyclical learning rates (CLR) were initially suggested by Smith (2015) and then revised by Smith (2017). CLR is a method for deciding the global learning rates when neural networks are being trained. It avoids the need to conduct multiple experiments and no additional computation is required when determining the best learning rates and schedule. Rather than just reducing the learning rate monotonically, this approach enables the learning rate to be adjusted between appropriate range values cyclically. Instead of using fixed learning rates, training by using cyclical learning rates will produce better accuracy. It requires fewer iterations and eliminates the necessity of adjusting the learning rate. The LR range test is used to begin training with a small learning rate. The learning rate gradually increased linearly over the pre-training process. This provides

relevant knowledge about the network's effect on training throughout various learning rates as well as information on the maximum learning rate.

One Cycle Policy (Smith, 2018), is a slight modification of CLR. It is a technique for determining the optimum learning rate during the neural network training. In the first half of the cycle, the learning rate gradually rises along with the simultaneous reduction in momentum, whereas in the second half of the cycle, the learning rate gradually falls along with the corresponding rise in momentum. Besides, Loshchilov and Hutter (2016) suggested a similar approach to CLR to speed up deep neural network training, which is stochastic gradient descent with restarts (SGDR), also known as Cosine Annealing. The learning rates decay along a cosine curve and then the learning rates will restart to their initial value at the end of the decay.

Xu et al. (2019) suggested a reinforcement learning-based framework, depending on the previous training histories. It has the ability to learn an adaptive learning rate schedule automatically. With this approach, the learning rate will adjust dynamically to recent training dynamics. Furthermore, Chandra and Sharma (2016) incorporated the principle of the Laplacian score with an adaptive learning rate; such updates involve algorithms that change the weights in mini-batches.

Moreover, Schaul et al. (2013) presented an approach to automatically adjust multiple learning rates. At each update and probably for every parameter, it determines an optimal learning rate, which optimizes the expected loss after the next update. This technique depends on local variations of gradients through samples. The learning rate will increase as well as decrease, making it an appropriate choice for non-stationary problems. The adaptive learning rate approach fully

removes the need to adjust the learning rate manually, or for its best value to be systematically searched.

## 2.5 Summary

In summary, the currently available strategies are either applying a fixed learning rate or a simplified adaptive updating scheme in every iteration. To address the research gap, i.e. manually search on the learning rate or fixed learning rate for all the components, we propose to develop a multiple adaptive learning rate strategy in order to achieve better performance and converge faster. The algorithm will adjust the learning rate in every iteration based on the approximated spectrum of the Hessian of the loss function.

## CHAPTER 3

### METHODOLOGY

#### 3.1 Mathematical Formulation of Multiple Adaptive Learning Rate Algorithms

To derive an updated scheme for the tuning parameter  $B_k$ , a restriction for components of  $B_k$  under some measures is imposed by minimizing the log-determinant norm and allowing them to satisfy the secant equation. Hence, for any positive diagonal matrix  $B_k$ , the solution is given by the updated  $B_{k+1}$ :

$$\min \text{tr}(B_{k+1}) - \ln(\det(B_{k+1})) \quad (3.1)$$

$$\text{s.t. } s_k^T B_{k+1} s_k = s_k^T y_k \quad (3.2)$$

where  $\text{tr}$  is the trace of a square matrix, it is defined to be the sum of elements on the main diagonal of the matrix;  $\det$  is the determinant of a matrix and

$$s_k = x_k - x_{k-1} \quad (3.3)$$

$$y_k = g_k - g_{k-1}. \quad (3.4)$$

Let  $B_{k+1} = \text{diag}(B_{k+1}^{(1)}, \dots, B_{k+1}^{(n)})$  and  $s_k = (s_k^{(1)}, \dots, s_k^{(n)})$ , the minimization problem (3.1) and (3.2) becomes

$$\min \left( \sum_{i=1}^n B_{k+1}^{(i)} - \ln \left( \prod_{i=1}^n B_{k+1}^{(i)} \right) \right) \quad (3.5)$$

$$s.t. \sum_{i=1}^n (s_k^{(i)})^2 B_{k+1}^{(i)} - s_k^T y_k = 0 \quad (3.6)$$

Next, we apply the Lagrange multiplier to the minimization problem (3.5) and (3.6) to get:

$$L(\rho) = \left( \sum_{i=1}^n B_{k+1}^{(i)} \right) - \ln \left( \prod_{i=1}^n B_{k+1}^{(i)} \right) + \rho \left[ \left( \sum_{i=1}^n (s_k^{(i)})^2 B_{k+1}^{(i)} \right) - s_k^T y_k \right] \quad (3.7)$$

where  $\rho$  is the Lagrange multiplier and can be approximated by using Newton-Raphson method.

Differentiate (3.7) with respect to  $B_{k+1}^{(i)}$  and set the derivatives to zero:

$$\frac{\partial L}{\partial B_{k+1}^{(i)}} = 1 - \frac{1}{B_{k+1}^{(i)}} + \rho (s_k^{(i)})^2, i = 1, 2, \dots, n \quad (3.8)$$

then gives

$$B_{k+1}^{(i)} = \frac{1}{1 + \rho (s_k^{(i)})^2}, i = 1, 2, \dots, n \quad (3.9)$$

By substituting (3.9) into constraint (3.6) gives:

$$F(\rho) = \sum_{i=1}^n \frac{(s_k^{(i)})^2}{1 + \rho (s_k^{(i)})^2} - s_k^T y_k \quad (3.10)$$

where the Lagrange multiplier  $\rho$  can be obtained by solving the objective function. This can be approximated by applying only one iteration of Newton-Raphson, with initial value of  $\rho = 0$ . When  $s_k^T s_k > s_k^T y_k$ , equation (3.10) has a unique positive solution and hence, the Lagrange multiplier  $\rho_k$  can be approximated by:

$$\rho \approx \frac{s_k^T s_k - s_k^T y_k}{\sum_{i=1}^n (s_k^{(i)})^4} \quad (3.11)$$

Lastly, the updating formula for  $B_{k+1}$  is

$$B_{k+1} = \text{diag}(B_{k+1}^{(1)}, \dots, B_{k+1}^{(n)}) \text{ for } s_k^T s_k > s_k^T y_k \quad (3.12)$$

In this project, since the diagonal matrix is used as the tuning parameter, therefore it is reasonable to use the log determinant norm. If the full-rank dense matrix is used, the log determinant norm becomes not practical due to the excessive computational operation of calculating the determinant of a full rank matrix. Additionally, the secant equation should be satisfied to ensure the boundedness of the updating formula.

There are three types of choices to update the case  $s_k^T s_k \leq s_k^T y_k$  in (3.12):

### Restoring

$$B_{k+1} = \begin{cases} \text{diag}(B_{k+1}^{(1)}, B_{k+1}^{(2)}, \dots, B_{k+1}^{(n)}) & \text{if } s_k^T s_k > s_k^T y_k \\ B_k & \text{otherwise.} \end{cases} \quad (3.13)$$

### Restarting

$$B_{k+1} = \begin{cases} \text{diag}(B_{k+1}^{(1)}, B_{k+1}^{(2)}, \dots, B_{k+1}^{(n)}) & \text{if } s_k^T s_k > s_k^T y_k \\ I & \text{otherwise.} \end{cases} \quad (3.14)$$

### Rescaling

$$B_{k+1} = \begin{cases} \text{diag}(B_{k+1}^{(1)}, B_{k+1}^{(2)}, \dots, B_{k+1}^{(n)}) & \text{if } s_k^T s_k > s_k^T y_k \\ \frac{s_k^T y_k}{s_k^T s_k} \cdot I & \text{otherwise.} \end{cases} \quad (3.15)$$

Restoring, restarting and rescaling are the options to preserve the positive definiteness of the updating formula  $B_k$ , to ensure every component in  $B_k$  is positive. All of the options have been tested in several datasets to analyze the dataset accordingly.

### 3.1.1 Multiple Adaptive Learning Rate Incorporated to SGD

The spectral parameter served as the multiple adaptive learning rate updating formula is incorporated to SGD, which was adapted from the approach suggested by Sim et al. (2019).

In order to preserve the positive definiteness of  $B_{k+1}$  whenever  $s_k^T s_k > s_k^T y_k$ , we will focus on using the restoring method. The restoring method will update the  $B_{k+1}$  to the previous  $B_k$  to preserve the positive definiteness of  $B_{k+1}$ .

The proposed method will adapt the learning rate in every iteration as well as in every single equation, i.e. every weight component and bias component. The proposed method adjusts the learning rate based on the approximated loss function's Hessian's spectrum. It uses the approximated inverse Hessian matrix of the loss function  $B^{-1}$  as the tuning parameter. The learning rate  $\mu$  in MAdaGrad is adapted by multiplying the inverse of  $B$ , which is  $\mu B^{-1}$ .  $\mu B^{-1}$  is a diagonal matrix that encapsulate the different learning rates in the algorithm. Since  $B$  is a diagonal matrix, it can be viewed as a vector with the same length  $l$  as parameters (weights and biases).

The algorithm of MAdaGrad is shown in algorithm 1.

### 3.1.2 Multiple Adaptive Learning Rate Incorporated to Adam

The Adam method has incorporated with the described updating formula of the multiple learning rate method to form a method named MAdaGrad Adam. The algorithm of MAdaGrad Adam is presented in



---

**Algorithm 1: MAdaGrad Algorithm**

---

- Step 0: Set  $k = 0$ . Given initial learning rate  $\mu = 0.001$  and  $B_0$  is a vector with all '1', velocity  $\alpha_0$  is a zero vector and momentum  $\rho = 0.9$
- Step 1: Compute  $y_k = g_k - g_{k-1}$  and  $s_k = \theta_k - \theta_{k-1}$  for  $k \geq 1$ .
- Step 2: Compute  $\omega = \frac{s_k^T s_k - s_k^T y_k}{\sum_{i=1}^n (s_k^{(i)})^4}$  for  $k \geq 1$ .
- Step 3: Compute  $B_k$  given by (3.13) for  $k \geq 1$ .
- Step 4: Compute  $\alpha_{k+1} = (\rho * \alpha_k) - \mu_k * B_k^{-1} * g_k$
- Step 5: Compute  $\theta_{k+1} = \theta_k - ((\rho * \alpha_{k+1}) - \mu_k * B_k^{-1} * g_k)$
- Step 6: If  $k = \text{maximum iteration}$ , stop. Else, set  $k = k + 1$  and go to step 1.
- 

Algorithm 2. The proposed method adjusts the learning rate based on the approximated inverse Hessian matrix of the loss function  $B^{-1}$  as the tuning parameter. The fixed learning rate in Adam is replaced by component-wise learning rate given by  $B^{-1}$  in this MAdaGrad Adam algorithm.

---

**Algorithm 2: MAdaGrad Adam Algorithm**

---

- Step 0: Set  $k = 0$ . Given initial learning rate  $\mu = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ ,  $\epsilon = 10^{-8}$ , first moment vector  $m_0$  and second moment vector  $v_0$  are initialized to 0 and time step  $t = 0$ .
- Step 1: Set  $t = t + 1$
- Step 2: Compute  $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$  and  $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
- Step 3: Compute  $\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$  and  $\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$
- Step 4: Compute  $y_t = g_t - g_{t-1}$  and  $s_t = \theta_t - \theta_{t-1}$
- Step 5: Compute  $B_k$  given by (3.13).
- Step 6: if  $t = 1$ ,  $\mu_t = \mu_0$ , else  $\mu_t = \mu_0 B_t^{-1}$
- Step 7:  $\theta_{t+1} = \theta_t - (\mu_t \times \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}})$
- Step 8: If  $k = \text{maximum iteration}$ , stop. Else, set  $k = k + 1$  and go to step 1.
- 

### 3.1.3 Convergence Analysis

In this section, we will show the convergence analysis of the proposed method according to the Assumption 1.

#### Assumption 1.

i The objective function  $f$  is twice continuously differentiable.

ii The level set  $D = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$  is convex.

iii There exist positive constants  $M_1$  and  $M_2$  such that

$$M_1 \|z\|^2 \leq z^T G(x) z \leq M_2 \|z\|^2 \quad (3.16)$$

for  $\forall z \in \mathbb{R}^n$  and  $\forall z \in D$ . This implies that the objective function  $f$  has a unique minimize  $x^*$  in  $D$ .

**Lemma 3.1.** To satisfy Assumption 1, assume that  $B_0 = I$ , where  $I$  is the  $n \times n$  identity matrix, and  $x_0$  is the starting point, so that  $f$  meets Assumption 1. The sequence  $\{\|B_k\|\}$  is then bounded by some positive constants  $c_1$  and  $c_2$ , where the  $B_{k+1}$  defined by (3.13).

*Proof.* Since  $\|B_0\|$  is bounded, we define  $\bar{G}$  as

$$\bar{G} = \int_0^1 \nabla^2 f(x_k + \tau s_k) d\tau. \quad (3.17)$$

According to the mean value theorem,

$$y_k = \bar{G} s_k \quad (3.18)$$

By replacing  $z$  by  $s_k$  in Assumption 1iii gives

$$M_1 \|s_k\|^2 \leq s_k^T y_k \leq M_2 \|s_k\|^2. \quad (3.19)$$

There are two cases in the updating formula of  $B_{k+1}$ :

Case 1: If  $s_k^T y_k < s_k^T s_k$ , for some  $k \geq 0$

$$B_{k+1}^{(i)} = \frac{1}{1 + \frac{s_k^T s_k - s_k^T y_k}{\sum_{i=1}^n (s_k^{(i)})^4} (s_k^{(i)})^2} \quad (3.20)$$

Let  $(s_k^{(M)})^2 = \max\{(s_k^{(1)})^2, \dots, (s_k^{(n)})^2\}$ .  $\|s_k\|^2 \leq n(s_k^{(M)})^2$  is derived from the assumption that  $s_k^T s_k = \sum_{i=1}^n (s_k^{(i)})^2$ . Thus,

$$\frac{s_k^T s_k - s_k^T y_k}{\sum_{i=1}^n (s_k^{(i)})^4} (s_k^{(i)})^2 \leq \frac{1 - M_1}{\sum_{i=1}^n (s_k^{(i)})^4} n (s_k^{(M)})^4 \leq n(1 - M_1). \quad (3.21)$$

It implies that

$$\frac{1}{1 + n(1 - M_1)} \leq B_{k+1}^{(i)} \leq 1. \quad (3.22)$$

where the bounds would also apply for  $B_0 = I$ , as  $\frac{s_k^T s_k - s_k^T y_k}{\sum_{i=1}^n (s_k^{(i)})^4} (s_k^{(i)})^2$  is nonnegative.

Case 2: If  $s_k^T y_k \geq s_k^T s_k$ , from Assumption 1iii gives

$$M_1 \leq B_{k+1}^{(i)} = \frac{s_k^T y_k}{s_k^T s_k} \leq M_2. \quad (3.23)$$

We combine equations (3.22) and (3.23) gives

$$c_1 = \min\left\{\frac{1}{1 + n(1 - M_1)}, M_1\right\} \leq B_k^{(i)} \leq \max\{1, M_2\} = c_2, \forall k \geq 0. \quad (3.24)$$

□

## 3.2 Implementation

### 3.2.1 Implementation of sklearn

Scikit-learn (sklearn) is the one of the most effective and robust library for machine learning in Python. Through a Python consistency interface, it offers a variety of effective tools for statistical modelling and machine learning, including clustering, classification, regression and dimensionality reduction. This library is based on NumPy, Matplotlib and SciPy, and it was written primarily in Python.

The algorithms have been implemented into Scikit-learn (sklearn) by modifying the source code of SGD and Adam. MAdaGrad and MAdaGrad Adam are added into the sklearn file `_multilayer_perceptron.py`.

```
1 from ._stochastic_optimizers import SGDOptimizer, AdamOptimizer,
   MAdaGrad, MAdaGrad_Adam
2
3 _STOCHASTIC_SOLVERS = ['sgd', 'adam', 'madagrad',
4 'madagrad_adam']
```

### Listing 3.1: `_multilayer_perceptron.py`

MAdaGrad and MAdaGrad Adam are added into `_fit_stochastic` function in `_multilayer_perceptron.py`.

```
1 if not incremental or not hasattr(self, '_optimizer'):
2     params = self.coefs_ + self.intercepts_
3
4     if self.solver == 'sgd':
5         self._optimizer = SGDOptimizer(params,
6 self.learning_rate_init, self.learning_rate, self.momentum, self
7         .nesterovs_momentum, self.power_t)
8     elif self.solver == 'adam':
9         self._optimizer = AdamOptimizer(params,
10 self.learning_rate_init, self.beta_1, self.beta_2,
11 self.epsilon)
12     elif self.solver == 'madagrad':
13         self._optimizer = MAdaGrad(params,
14 self.learning_rate_init, self.learning_rate, self.momentum, self
15         .nesterovs_momentum, self.power_t)
16     elif self.solver == 'madagrad_adam':
17         self._optimizer = MAdaGrad_Adam(params,
18 self.learning_rate_init, self.beta_1, self.beta_2,
19 self.epsilon)
```

### Listing 3.2: `_fit_stochastic` function

The sklearn code is also modified in the file `_stochastic_optimizers.py`.

### 3.2.2 SGD

In the python machine learning module, sklearn with version 0.24.2, the stochastic gradient descent method consists of three types of learning rate schedules:

#### Constant

The constant learning rate schedule is the default learning rate schedule for the SGD method. The same learning rate is used throughout the training:

$$\mu^{(t)} = \mu_0. \quad (3.25)$$

#### Invscaling

The inverse scaling updating formula is given by:

$$\mu^{(t)} = \frac{\mu_0}{t^{\text{power}_t}}. \quad (3.26)$$

where  $\mu_0$  is the initial learning rate and the default value of `power_t` is 0.5.

#### Adaptive

Like the constant learning rate case, first, let the learning rate to be a constant

$$\mu^{(t)} = \mu_0. \quad (3.27)$$

Then for every two consecutive epochs that fail to show the reduction in the training loss by a fixed tolerance  $tol = 10^{-3}$  or fail to increase the validation score by  $tol$ , the current learning rate is divided by 5, and the algorithm does not stop. The algorithm stops when the learning rate goes below  $10^{-6}$  and when the learning rate has no improvement for 10 executive times.

Listing 3.3 is the updating formula for SGD in sklearn.

```
1 updates = [self.momentum * velocity - self.learning_rate * grad
             for velocity, grad in zip(self.velocities, grads)]
```

### Listing 3.3: SGD Updating Formula

The learning rate  $\alpha$  in MAdaGrad is adapted by multiplying the inverse of  $B$ , which is  $\alpha B^{-1}$ .  $B^{-1}$  is a vector with the same length  $q$  as parameters (weights and biases). The maximum and minimum values of  $\alpha B^{-1}$  has captured in every iteration. The values have collected from 30 random states of data in 10 iterations. The average of the maximum and minimum values have been used as the default learning rate in SGD, which are the SGD(max) and SGD(min).

### 3.2.3 MAdaGrad

A new class MAdaGrad is added to sklearn documentation.

Some new initial parameters are added to the MAdaGrad optimizer.

```
1 # previous parameters
2 self.prev_params = None
3 # previous gradient
4 self.prev_gradient = None
5 # initial B
6 self.B = []
7 for i in range(len(self.params)):
8     self.B.append(np.ones_like(self.params[i]))
9 # initial omega
10 self.omega = [None] * (len(self.params))
```

```

11 # number of hidden layers
12 self.no_hlayers = len(self.params) // 2

```

### Listing 3.4: Initial parameters

Listing 3.5 is the function to compute  $s$ .

```

1 def compute_s(self):
2     # difference between current parameters and previous
3     # parameters
4     s = [a1 - a2 for a1, a2 in zip(self.params,
5 self.prev_params)]
6     return s

```

### Listing 3.5: Compute $s$

Listing 3.6 is the function to compute  $y$ .

```

1 def compute_y_diff(self, grads):
2     # difference between current gradient and previous gradient
3     y_diff = [a1 - a2 for a1, a2 in zip(grads,
4 self.prev_gradient)]
5     return y_diff

```

### Listing 3.6: Compute $y$

Listing 3.7 is the function to compute  $\omega$ .

```

1 def compute_omega(self, s, y_diff):
2     for i in range(self.no_hlayers * 2):
3         if np.sum([a ** 4 for a in s[i]]) == 0:
4             self.omega[i] = 0
5         else:
6             self.omega[i] = (np.sum([a1 * a2 for a1, a2 in zip(s[
7 in zip(s[i], y_diff[i])])) - np.sum([a1 * a2 for a1, a2

```

### Listing 3.7: Compute $\omega$

Listing 3.8 is the function to compute  $B$ .

```

1 def compute_B(self, grads):
2     prev_params = self.prev_params
3     prev_gradient = self.prev_gradient
4     B = self.B
5
6     if prev_params is None and prev_gradient is None:
7         # gradient descent
8         pass
9     else:
10        # compute s and y
11        s = self.compute_s()
12        y_diff = self.compute_y_diff(grads)
13
14        # compute omega
15        self.compute_omega(s, y_diff)
16
17        # compute B
18        for i in range(self.no_hlayers * 2):
19            c1 = np.sum([a1 * a2 for a1,a2
20 in zip(s[i], s[i])])
21            c2 = np.sum([a1 * a2 for a1,a2
22 in zip(s[i], y_diff[i])])
23
24            if c1 > c2:
25                B[i] = 1 / (1 + self.omega[i] * (s[i] ** 2))
26            else:
27                B[i] = B[i]

```

**Listing 3.8: Compute  $B$**

$B^{-1}$  has been computed and incorporated into SGD to form the updating formula for MAdaGrad.

```

1 self.compute_B(grads)
2 inverse_B = [1.0 / _B for _B in self.B]
3
4 updates = [self.momentum * velocity - lr * inv_B * grad

```



```

5         for velocity, grad, inv_B in zip(self.velocities
6         , grads, inverse_B)]

```

### Listing 3.9: MAdaGrad Updating Formula

A deepcopy is made on the current parameters and gradients for the next iteration.

```

1 self.prev_params = copy.deepcopy(self.params)
2 self.prev_gradient = copy.deepcopy(grads)

```

### 3.2.4 MAdaGrad Adam

Listing 3.10 is the updating formula for Adam.

```

1 self.t += 1
2 self.ms = [self.beta_1 * m + (1 - self.beta_1) * grad
3         for m, grad in zip(self.ms, grads)]
4 self.vs = [self.beta_2 * v + (1 - self.beta_2) * (grad ** 2)
5         for v, grad in zip(self.vs, grads)]
6 self.learning_rate = (self.learning_rate_init *
7         np.sqrt(1 - self.beta_2 ** self.t) /
8         (1 - self.beta_1 ** self.t))
9 updates = [-self.learning_rate * m / (np.sqrt(v)
10 + self.epsilon)
11         for m, v in zip(self.ms, self.vs)]

```

### Listing 3.10: Adam Updating Formula

$B^{-1}$  has been computed and incorporated into Adam to form the updating formula for MAdaGrad Adam.

```

1 if self.t == 1:
2     self.learning_rate = (self.learning_rate_init
3 * np.sqrt(1 - self.beta_2 ** self.t) / (1 - self.beta_1 ** self.
4     t))
5     updates = [-self.learning_rate * m / (np.sqrt(v)
6 + self.epsilon) for m, v in zip(self.ms, self.vs)]
7 else:

```

```

7     self.learning_rate = [self.learning_rate_init * inv_B
8     for inv_B in inverse_B]
9
10    updates = [-lr * m / (np.sqrt(v) + self.epsilon)
11               for m, v, lr in zip(self.ms, self.vs
12               , self.learning_rate)]

```

**Listing 3.11: MAdaGrad Adam Updating Formula**

A deepcopy is made on the current parameters and gradients for the next iteration.

```

1 self.prev_params = copy.deepcopy(self.params)
2 self.prev_gradient = copy.deepcopy(grads)

```

**Listing 3.12: Storing previous parameters and gradients**

### 3.3 Benchmark

The software Python 3.8.5 is downloaded on MSI GL62M 7RDX with processor core i7 and RAM 8GB. By using the performance profile of Dolan and Moré (2002), the performance of the methods MAdaGrad, SGD(constant), SGD(adaptive) and SGD(adaptive) can be evaluated clearly. The performance on problem  $P$  by solver  $\hat{s}$  is defined as:

$$\mathbb{P}(\hat{t} \leq \tau) = \frac{1}{|P|} \text{size}\{\hat{p} \in P : \hat{t}_{\hat{p}, \hat{s}} \leq \tau\},$$

where the function  $\mathbb{P}(\hat{t} \leq \tau)$  is the cumulative distribution function for the performance ratio,  $P$  is a set of test problems,  $|P|$  denotes the cardinality of  $P$  and  $\hat{t}_{\hat{p}, \hat{s}}$  represents the performance ratio within a factor  $\tau$  which is a real number

$$\hat{t}_{\hat{p}, \hat{s}} = \frac{m_{\hat{p}, \hat{s}}}{\min m_{\hat{p}, \hat{s}} : \hat{s} \in S'}$$

where  $m_{\hat{p},s}$  represents the performance measure of interest. It is obtained for each pair  $(\hat{p},\hat{s})$  of solver  $s$  in a set  $S$  of optimization solvers and problem  $P$  in a set  $P$  of test problems. The values of  $\mathbb{P}(\hat{t} \leq \tau)$  are bounded between 0 and 1. The higher the value of  $\mathbb{P}(\hat{t} \leq \tau)$  indicates better performance. We perform the comparisons on SGD family and Adam family, and discuss separately.

## CHAPTER 4

### NUMERICAL RESULTS AND DISCUSSIONS

#### 4.1 Datasets

To illustrate the performance of the proposed method, we apply the proposed method to the following datasets:

1. MNIST handwritten digits datasets: divided into 60000 training and 10000 testing images. Each image consists of 28x28 pixels (i.e. 784 features) gray scale images of handwritten single digits between 0 and 9. Each feature represents only one pixel's intensity i.e. from 0(white) to 255(black).
2. Breast Cancer datasets: consists of 2 classes ('malignant' and 'benign'), 30 attributes and 569 instances.
3. Wine datasets: consists of three types of wine ('class 0', 'class 1', 'class 2'), 13 attributes and represented in the 178 samples.
4. Abalone datasets: consists of 29 classes, 8 attributes and 4177 samples.

The datasets have been split as follows: 25% of the actual data becomes the testing set and 75% of the actual data becomes the training set.

**Table 4.1: Attribute Information of Breast Cancer Datasets**

Attribute Name	Data Type	Mean	Standard Deviation
Radius Mean	Continuous	14.127	3.521
Texture Mean	Continuous	19.290	4.297
Perimeter Mean	Continuous	91.970	24.278
Area Mean	Continuous	654.900	351.605
Smoothness Mean	Continuous	0.096	0.014
Compactness Mean	Continuous	0.104	0.053
Concavity Mean	Continuous	0.089	0.080
Concave Points Mean	Continuous	0.049	0.039
Symmetry Mean	Continuous	0.181	0.027
Fractal Dimension Mean	Continuous	0.063	0.007
Radius Error	Continuous	0.405	0.277
Texture Error	Continuous	1.217	0.551
Perimeter Error	Continuous	2.866	2.020
Area Error	Continuous	40.337	45.451
Smoothness Error	Continuous	0.007	0.003
Compactness Error	Continuous	0.025	0.018
Concavity Error	Continuous	0.032	0.030
Concave Points Error	Continuous	0.012	0.006
Symmetry Error	Continuous	0.021	0.008
Fractal Dimension Error	Continuous	0.004	0.003
Worst Radius	Continuous	16.270	4.829
Worst Texture	Continuous	25.680	6.141
Worst Perimeter	Continuous	107.260	33.573
Worst Area	Continuous	880.600	568.856
Worst Smoothness	Continuous	0.132	0.023
Worst Compactness	Continuous	0.254	0.157
Worst Concavity	Continuous	0.272	0.208
Worst Concave Points	Continuous	0.115	0.066
Worst Symmetry	Continuous	0.290	0.062
Worst Fractal Dimension	Continuous	0.084	0.018

## 4.2 Experimental Setup

We apply the MAdaGrad and MAdaGrad Adam methods as the optimization methods to train the neural network. The comparison is made between the MAdaGrad method and the three other methods. The following methods are taken into consideration:

1. Multiple Adaptive Learning Rate Method (MAdaGrad)
2. SGD with constant learning rate schedule (SGD(constant))

**Table 4.2: Attribute Information of Wine Datasets**

Attribute Name	Data Type	Mean	Standard Deviation
Alcohol	Continuous	13.000	0.800
Malic acid	Continuous	2.340	1.120
Ash	Continuous	2.360	0.270
Alcalinity of ash	Continuous	19.500	3.300
Magnesium	Continuous	99.700	14.300
Total phenols	Continuous	2.290	0.630
Flavanoids	Continuous	2.030	1.000
Nonflavanoid phenols	Continuous	0.360	0.120
Proanthocyanins	Continuous	1.590	0.570
Color intensity	Continuous	5.100	2.300
Hue	Continuous	0.960	0.230
OD280/OD315	Continuous	2.610	0.710
Proline	Continuous	746.000	315.000

**Table 4.3: Attribute Information of Abalone Datasets**

Attribute Name	Data Type	Mean	Standard Deviation
Sex	Nominal	-	-
Length	Continuous	0.524	0.120
Diameter	Continuous	0.408	0.099
Height	Continuous	0.140	0.042
Whole Weight	Continuous	0.829	0.490
Shucked Weight	Continuous	0.359	0.222
Viscera Weight	Continuous	0.181	0.110
Shell Weight	Continuous	0.239	0.139
Rings	Integer	-	-

3. SGD with invscaling learning rate schedule (SGD(invscaling))
4. SGD with adaptive learning rate schedule (SGD(adaptive))

The default learning rate  $\mu$  is 0.001. The vector  $B_0$  is initialized to a vector with all '1' with length  $l$ . Thirty different random states with different hidden layer sizes have been used to plot the profiling graphs.

The loss function used is the cross entropy loss:

$$-\sum_{i=1}^N y_i \cdot \log \hat{y}_i$$

where  $N$  is the size of output layer, i.e. the number of classes in the datasets.  $y$  is the actual value and  $\hat{y}$  is the predicted value of the

classification problems. The datasets listed in section 4.1 will be applied into machine learning prediction. Different datasets give different value of  $y$ :

1. MNIST handwritten digits datasets: handwritten single digits between 0 and 9.
2. Breast cancer datasets: prognosis 'malignant' or prognosis 'benign'.
3. Wine datasets: wine type '1', '2' and '3'.
4. Abalone datasets: class number of rings.

There are two termination criteria used to observe the performance of the method, which are the loss values and number of iterations. The maximum number of iterations has been set to 500 and the loss values are used to profile the performance of the method. The hidden layer sizes used for the profiling graphs are  $(1,100)$ ,  $(2,100)$ ,  $(3,100)$ ,  $(4,100)$ ,  $(1,1000)$  and  $(1,2000)$ , where  $(p,q)$  denotes the hidden layer sizes of the neural network, with  $p$  denoting the number of layer in the neural network and  $q$  denoting the number of neurons in each layer. For example,  $(3,100)$  is a neural network containing 3 layers with 100 neurons in each layer. The second termination condition is the number of iterations. If the number of iterations reaches the upper limit of 10000, then it will be considered as fail to converge. The hidden layer sizes used for the profiling graphs are  $(1,100)$ ,  $(2,100)$ ,  $(1,1000)$ ,  $(1,2000)$  and  $(1,5000)$ .

### **4.3 Comparison between MAdaGrad and SGD - Loss values**

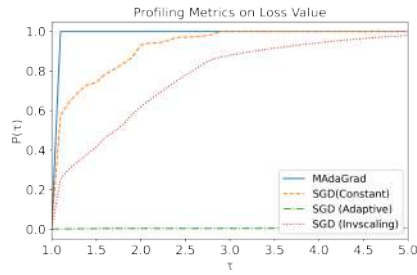
#### **4.3.1 MNIST handwritten digits datasets**

Figures 4.1 and 4.2 are the profiling graphs of loss value in different hidden layer sizes for the methods MAdaGrad, SGD(adaptive) and

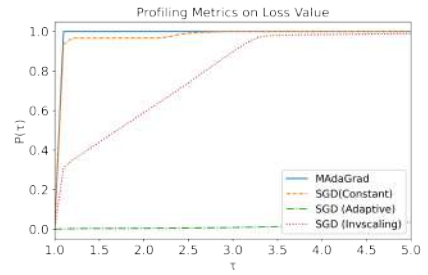
SGD(invscaling). The profiling graphs for batch sizes 200 and 500 are shown. From the figures, MAdaGrad performs the best among these methods, followed by SGD(Adaptive). MAdaGrad gives the best results for every random state, thus the  $\mathbb{P}(t \leq \tau)$  for MAdaGrad increases to 1 at the beginning stage for every hidden layer size. SGD(invscaling) method shows poor performances among these methods. The loss values decrease slowly and it has the largest loss value in the 500<sup>th</sup> iterations among these methods. As the number of neurons increases, i.e., hidden layer sizes = (1, 100), (1,1000), (1,2000), the value of  $\mathbb{P}(t \leq \tau)$  grows faster as  $\tau$  increases.

MAdaGrad performs better than SGD(adaptive) but does not outperform it as the number of neurons increases. The training weights are interdependent in a neural network, however, in MAdaGrad, the weight components are optimized separately. As the number of neurons increases, more weight components are interconnected and thus lead to little difference in cumulative probability in the optimization of MAdaGrad. On the other hand, as the number of layers increases, MAdaGrad still shows the best performance among these methods. SGD(constant) is not included in the profiling graph since the results of SGD(constant) and SGD(adaptive) are almost the same, the line of these two methods are overlapping due to the small difference in loss values. The other datasets, which are the breast cancer dataset, wine dataset and abalone dataset, exhibit similar patterns as MNIST digit datasets. SGD(constant) and SGD(adaptive) give similar results because both are the same when there are no two consecutive epochs that fail to show a reduction or fail to increase the validation score, as mentioned in section 3.2.2.

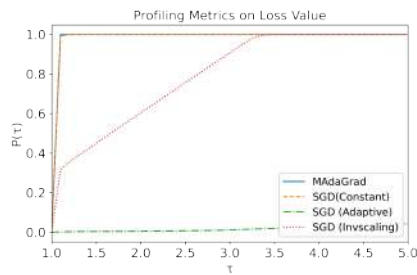




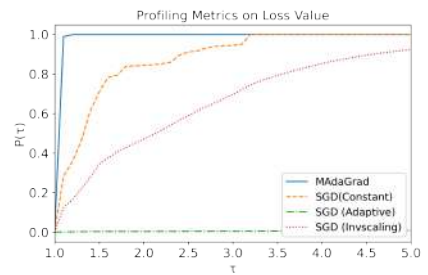
(a) Hidden layer size (100,)



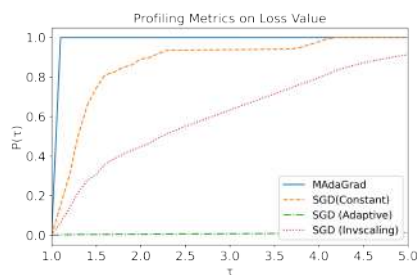
(b) Hidden layer size (1000,)



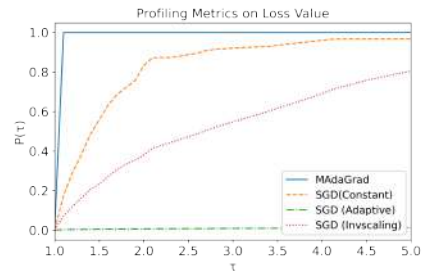
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

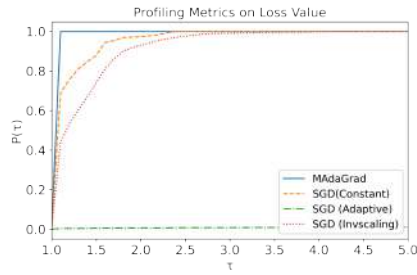


(e) Hidden layer size (100,100,100)

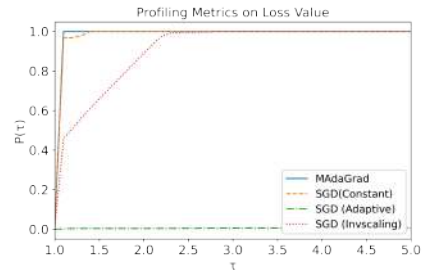


(f) Hidden layer size (100,100,100,100)

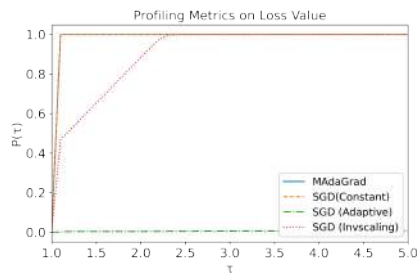
Figure 4.1: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (MNIST Dataset).



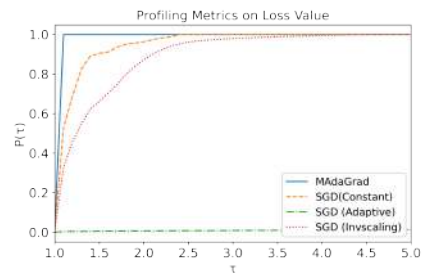
(a) Hidden layer size (100,)



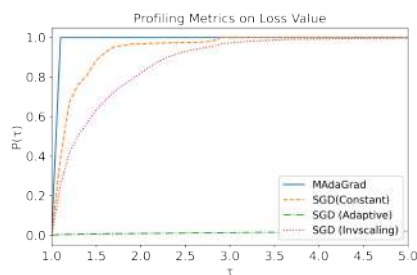
(b) Hidden layer size (1000,)



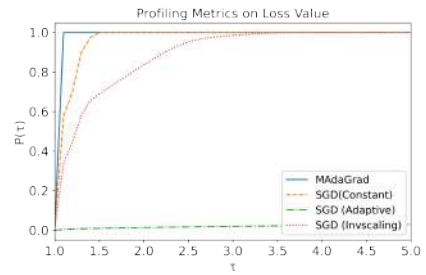
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)

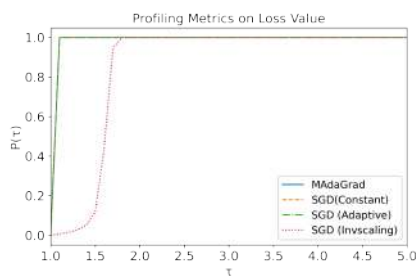


(f) Hidden layer size (100,100,100,100)

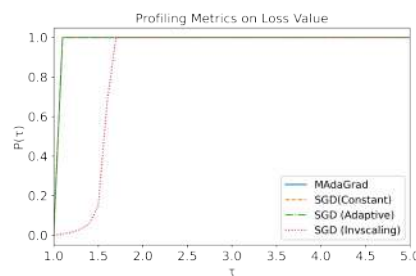
Figure 4.2: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (MNIST Dataset).

### 4.3.2 Abalones datasets

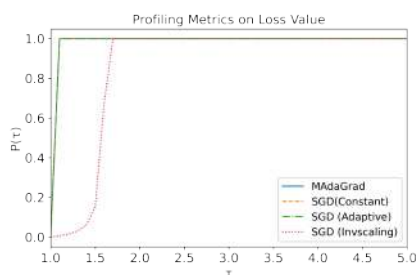
Profiling graphs for the batch sizes of 200 and 500 are shown in figures 4.3 and 4.4. From the figures, although MAdaGrad and SGD(adaptive) gives a similar results, MAdaGrad still performs slightly better than SGD(adaptive). SGD(invscaling) performs the worst among these methods.



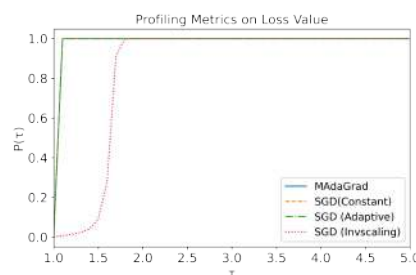
(a) Hidden layer size (100,)



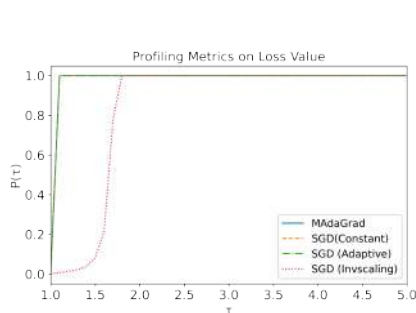
(b) Hidden layer size (1000,)



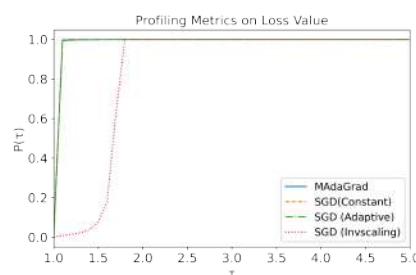
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)

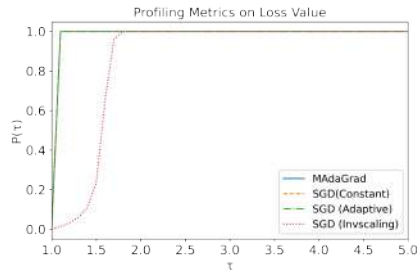


(f) Hidden layer size (100,100,100,100)

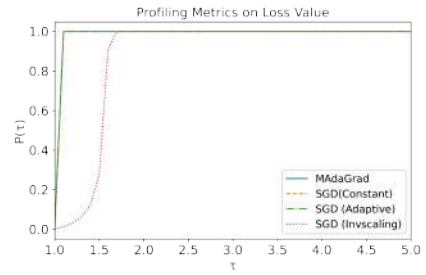
Figure 4.3: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Abalone Dataset).

### 4.3.3 Breast cancer datasets

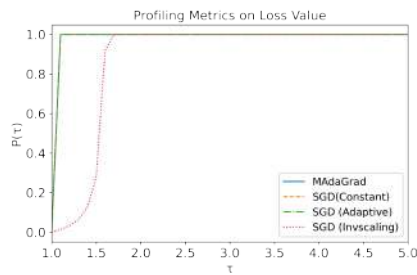
Figures 4.5 and 4.6 are the profiling graphs of loss value for batch sizes 200 and 500. As the batch size increase from 200 to 500, the



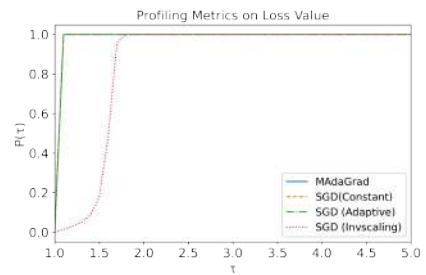
(a) Hidden layer size (100,)



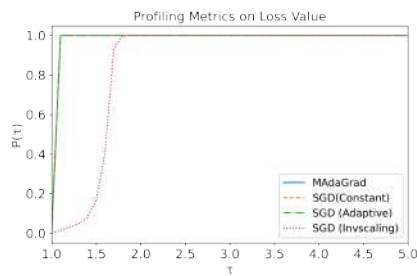
(b) Hidden layer size (1000,)



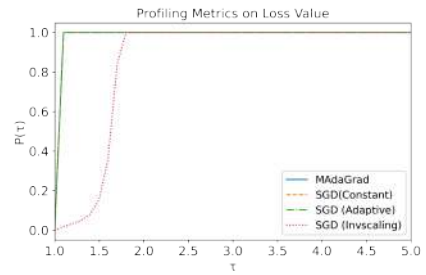
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



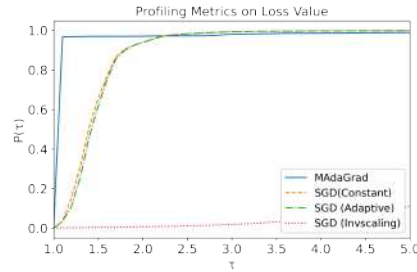
(e) Hidden layer size (100,100,100)



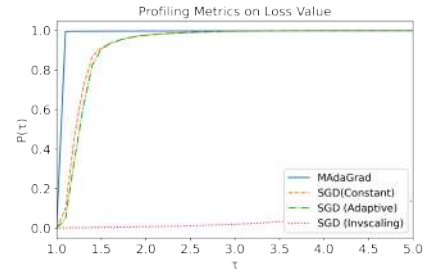
(f) Hidden layer size (100,100,100,100)

Figure 4.4: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Abalone Dataset).

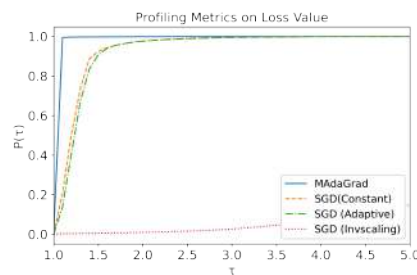
distinction between MAdaGrad and adaptive SGD diminishes when using large batch sizes, as such batch sizes can cause models to become trapped in local minima.



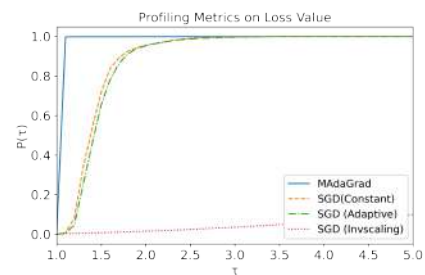
(a) Hidden layer size (100,)



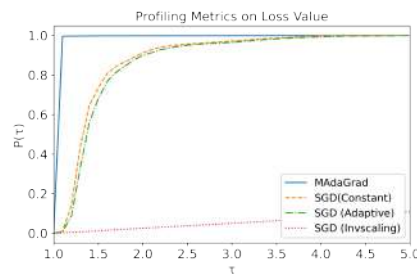
(b) Hidden layer size (1000,)



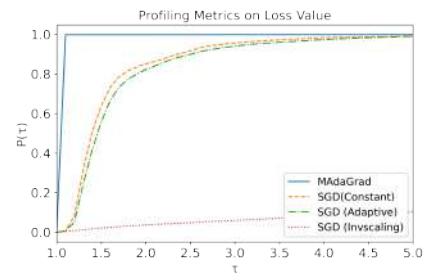
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)

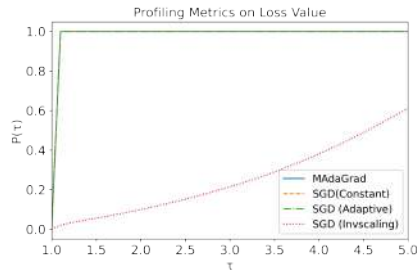


(f) Hidden layer size (100,100,100,100)

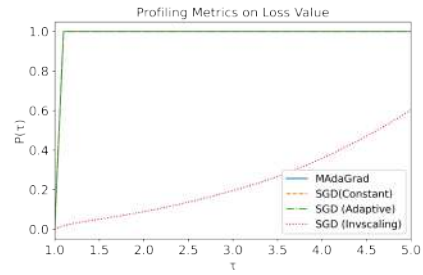
Figure 4.5: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Breast Dataset).

#### 4.3.4 Wine datasets

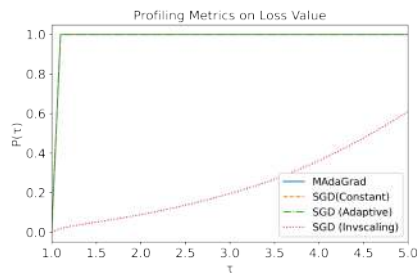
Figures 4.7 and 4.8 are the profiling graphs of loss value in different batch sizes and hidden layer sizes. Wine dataset exhibits similar patterns as MNIST digit datasets.



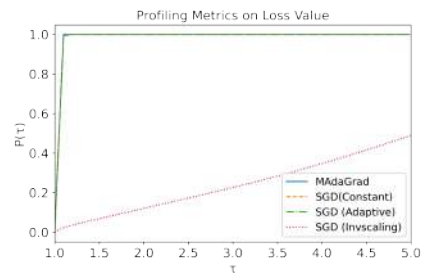
(a) Hidden layer size (100,)



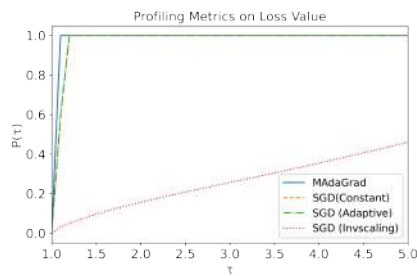
(b) Hidden layer size (1000,)



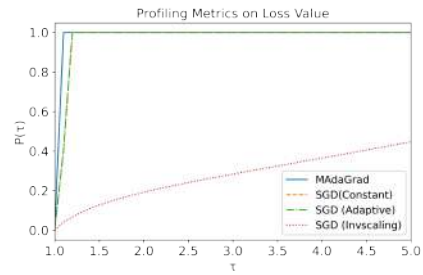
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

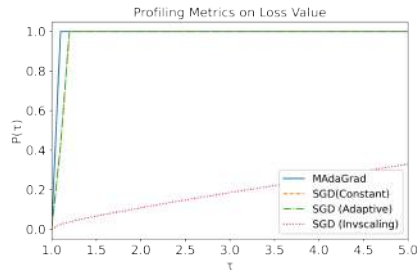


(e) Hidden layer size (100,100,100)

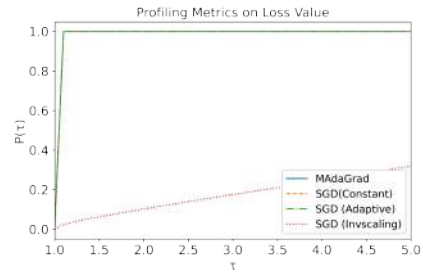


(f) Hidden layer size (100,100,100,100)

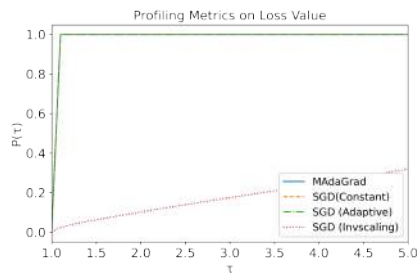
Figure 4.6: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Breast Dataset).



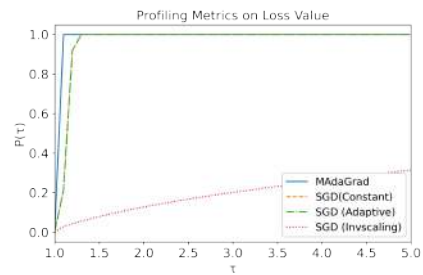
(a) Hidden layer size (100,)



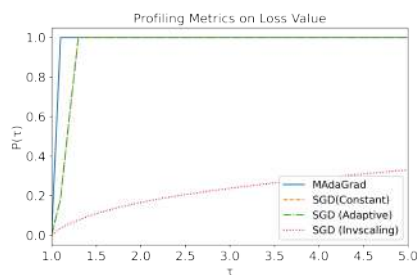
(b) Hidden layer size (1000,)



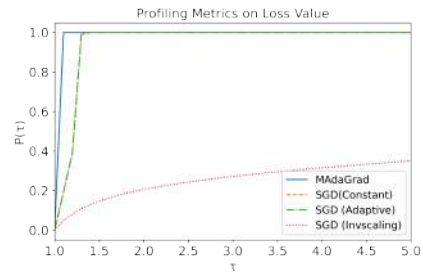
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

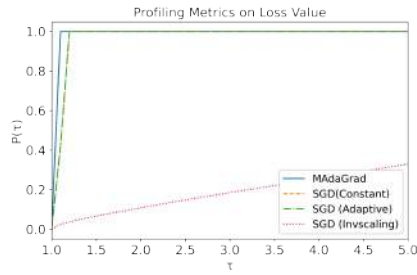


(e) Hidden layer size (100,100,100)

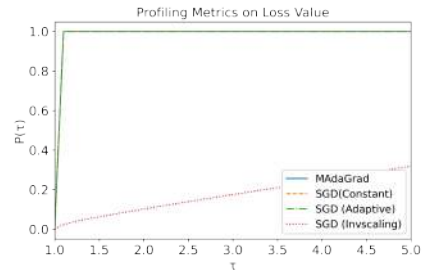


(f) Hidden layer size (100,100,100,100)

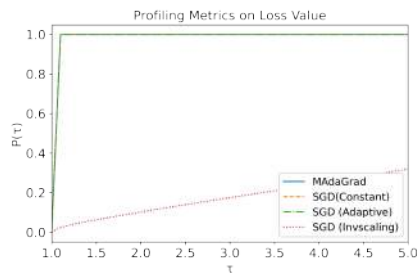
Figure 4.7: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Wine Dataset).



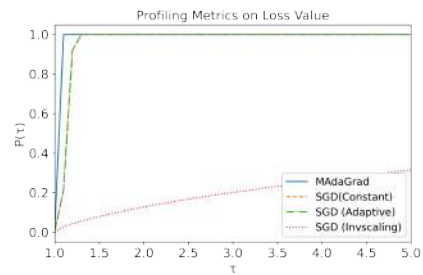
(a) Hidden layer size (100,)



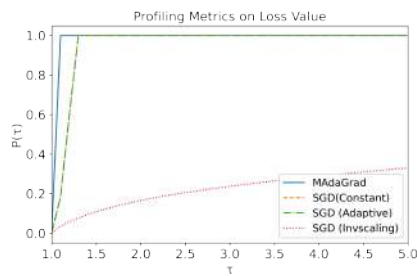
(b) Hidden layer size (1000,)



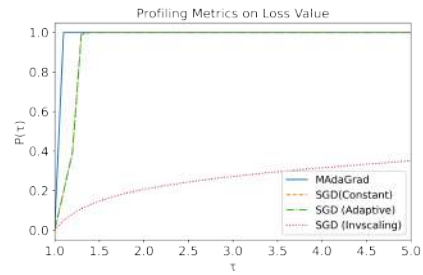
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)



(f) Hidden layer size (100,100,100,100)

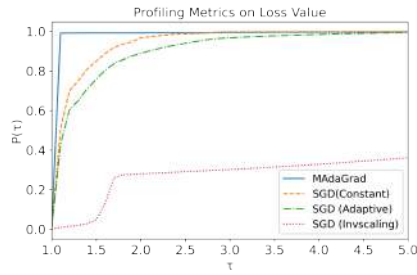
Figure 4.8: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Wine Dataset).



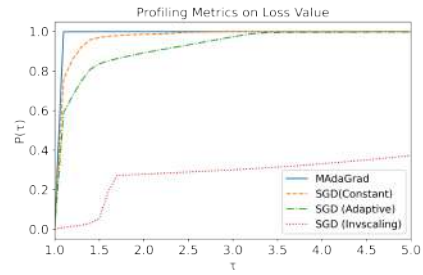
### 4.3.5 Combined Datasets

Figures 4.9 and 4.10 are the profiling graphs of loss value for the methods MAdaGrad, SGD(constant), SGD(adaptive) and SGD(invscaling). The results are performed in batch sizes 200 and 500 for different hidden layer sizes. From the figures, MAdaGrad performs the best among these methods, followed by SGD(Constant) and SGD(Adaptive). MAdaGrad gives the best results for every random state, thus the  $\mathbb{P}(\hat{t} \leq \tau)$  for MAdaGrad increases to 1 at the beginning stage for every hidden layer size. SGD(invscaling) method shows poor performances among these methods. The loss values decrease slowly and it has the largest loss value in the 500<sup>th</sup> iterations among these methods.

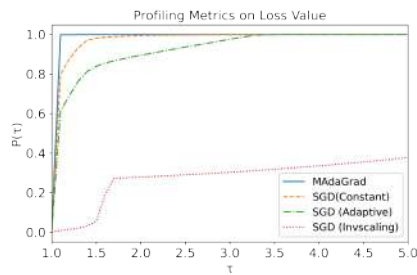
MAdaGrad gives the best results for every random state, thus the  $\mathbb{P}(\hat{t} \leq \tau)$  for MAdaGrad increases to 1 at the beginning stage for every hidden layer size. The performance of the SGD(constant) method is improved as the number of neurons increases, i.e., hidden layer sizes = (100,), (1000,), (2000,). MAdaGrad performs better than SGD(constant) but does not outperform it as the number of neurons increases. The training weights are interdependent in a neural network, however, in MAdaGrad, the weight components are optimized separately. As the number of neurons increases, more weight components are interconnected and show similar results for the optimization of MAdaGrad and SGD. On the other hand, as the number of layers increases, MAdaGrad gives the best performance among these methods.



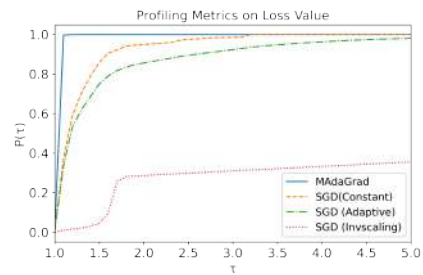
(a) Hidden layer size (100,)



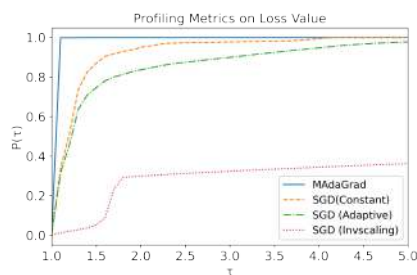
(b) Hidden layer size (1000,)



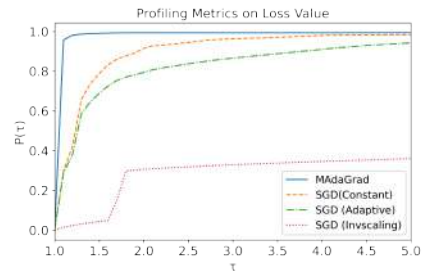
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

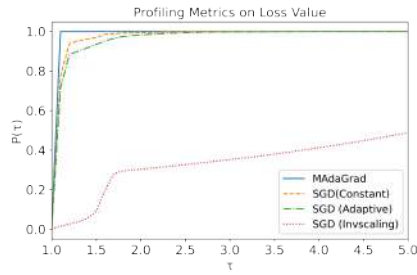


(e) Hidden layer size (100,100,100)

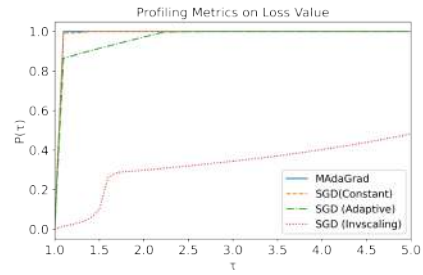


(f) Hidden layer size (100,100,100,100)

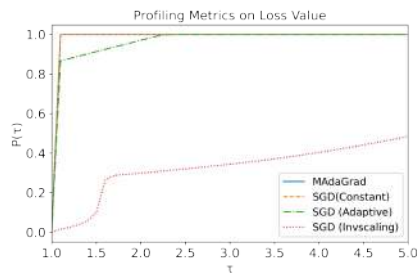
Figure 4.9: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (Combined Dataset).



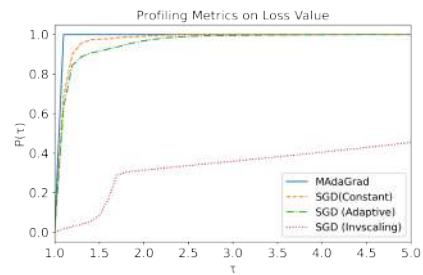
(a) Hidden layer size (100,)



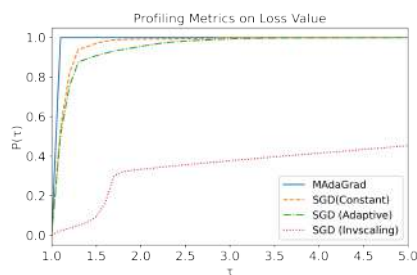
(b) Hidden layer size (1000,)



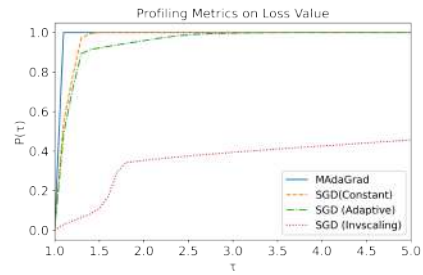
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)



(f) Hidden layer size (100,100,100,100)

Figure 4.10: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 500 (Combined Dataset).

## 4.4 Comparison between MAdaGrad Adam and Adam - Loss values

### 4.4.1 MNIST handwritten digits datasets

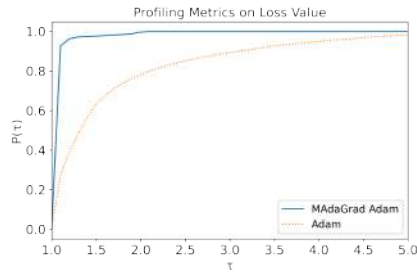
Figures 4.11 and 4.12 are the performance profiling graphs for the methods MAdaGrad Adam and Adam in terms of loss values for the batch sizes of 200 and 500. They give a similar pattern according to different hidden layer sizes. In the hidden layer sizes (100,), (100,100), (100,100,100) and (100,100,100,100), MAdaGrad Adam performs better than Adam. The performances of MAdaGrad Adam are not affected as the number of layers increases. In the hidden layer sizes (1000,) and (2000,), MAdaGrad Adam gives poorer performances than Adam. As the number of neurons increased, MAdaGrad Adam shows no significant improvement compared to Adam. As mentioned before, the weight components are interconnected, and when the number of neurons increases, will result in little cumulative probability difference in the optimization of MAdaGrad Adam.

### 4.4.2 Abalone datasets

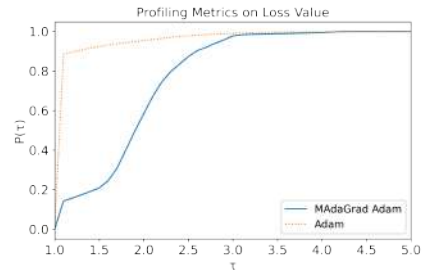
Figures 4.13 and 4.14 are the performance profiling graphs for different hidden layer sizes. As the number of neurons increases, MAdaGrad Adam still performs slightly better than Adam. On the other hand, when the number of layers increases, the performance of MAdaGrad Adam remains, and the performance of Adam declines, thus MAdaGrad Adam provides a better result than Adam.

### 4.4.3 Breast cancer datasets

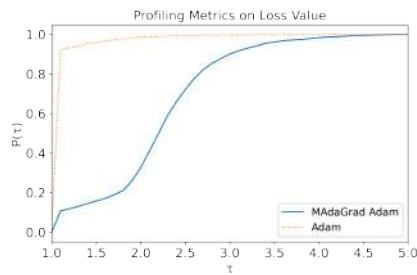
Figures 4.15 and 4.16 indicate that although the difference of  $P(\tau)$  between these two methods decrease as number of neurons and layers increase, MAdaGrad Adam still outperforms Adam.



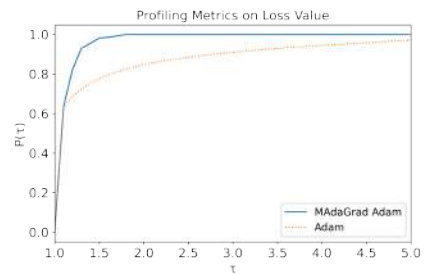
(a) Hidden layer size (100,)



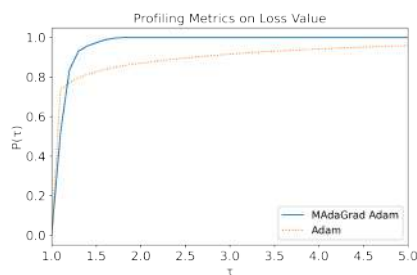
(b) Hidden layer size (1000,)



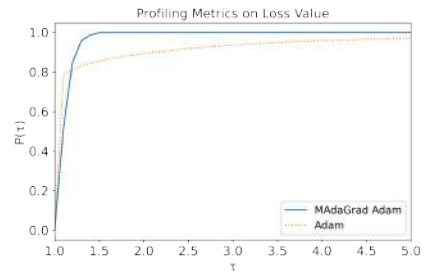
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

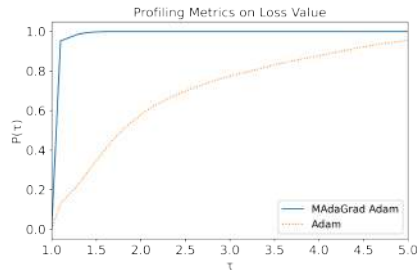


(e) Hidden layer size (100,100,100)

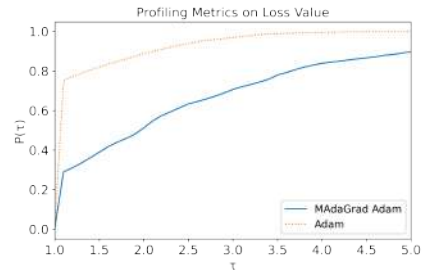


(f) Hidden layer size (100,100,100,100)

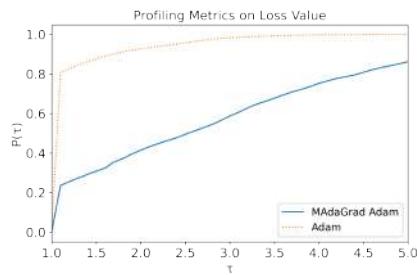
Figure 4.11: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (MNIST Dataset).



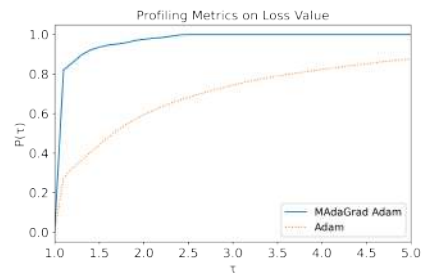
(a) Hidden layer size (100,)



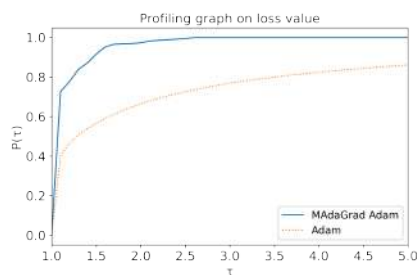
(b) Hidden layer size (1000,)



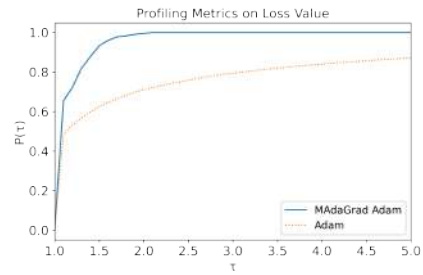
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

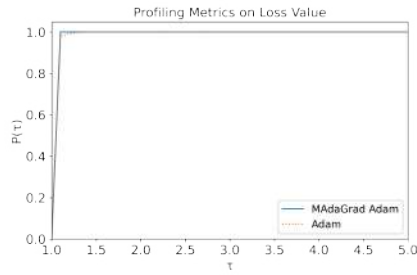


(e) Hidden layer size (100,100,100)

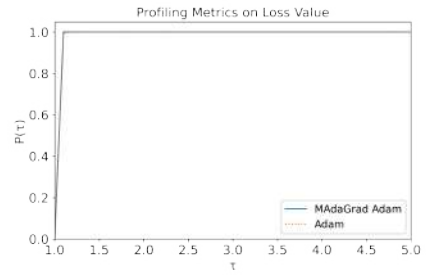


(f) Hidden layer size (100,100,100,100)

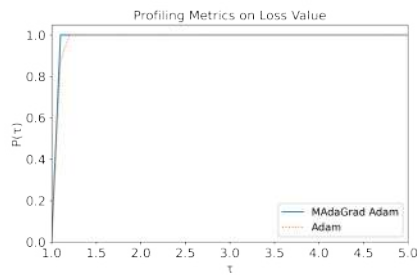
Figure 4.12: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (MNIST Dataset).



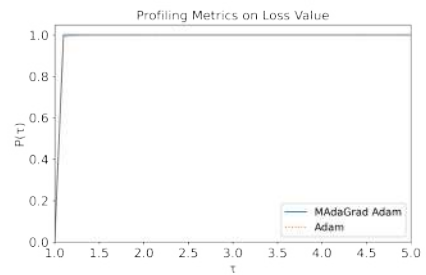
(a) Hidden layer size (100,)



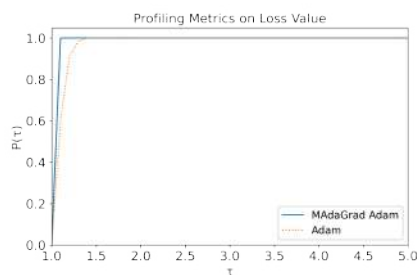
(b) Hidden layer size (1000,)



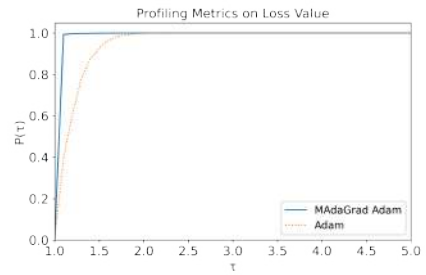
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

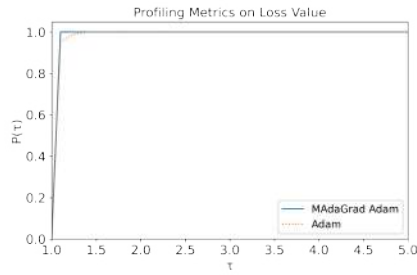


(e) Hidden layer size (100,100,100)

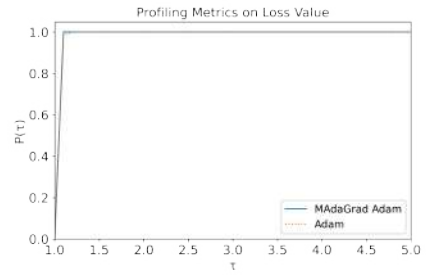


(f) Hidden layer size (100,100,100,100)

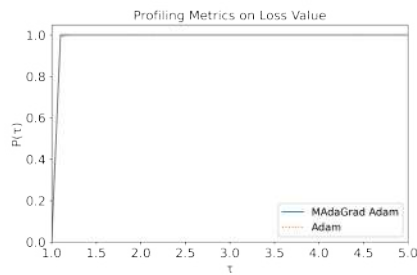
Figure 4.13: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Abalone Dataset).



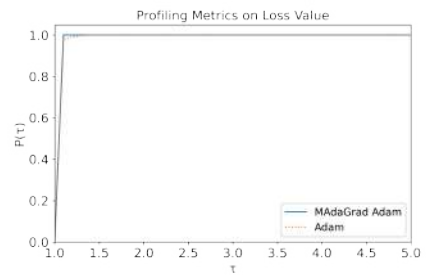
(a) Hidden layer size (100,)



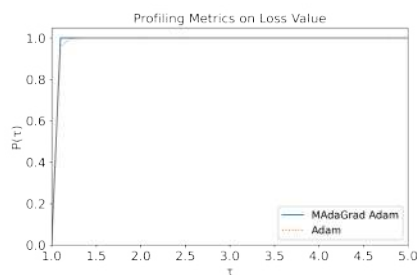
(b) Hidden layer size (1000,)



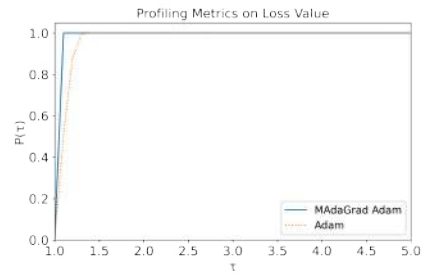
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



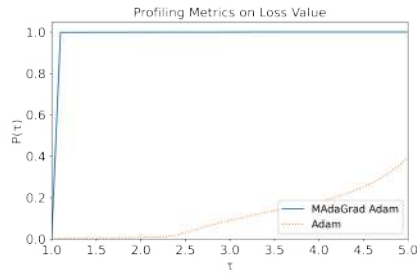
(e) Hidden layer size (100,100,100)



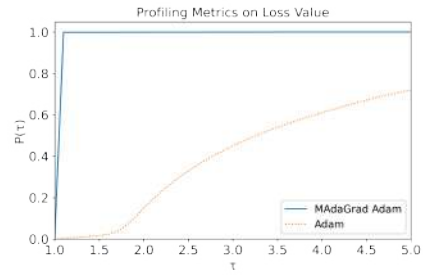
(f) Hidden layer size (100,100,100,100)

Figure 4.14: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Abalone Dataset).

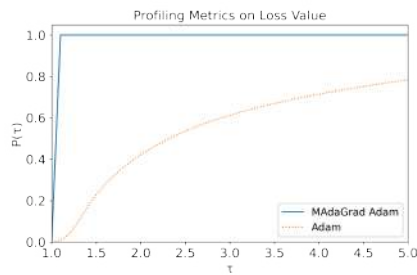




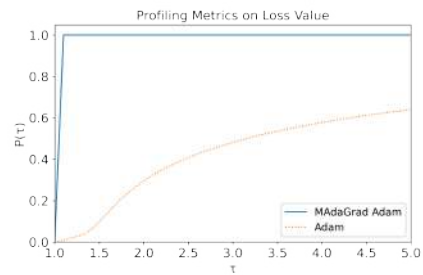
(a) Hidden layer size (100,)



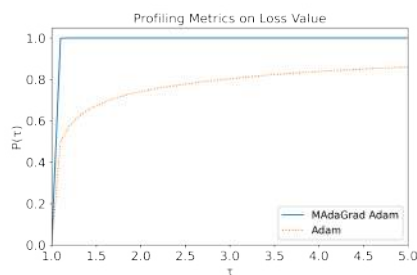
(b) Hidden layer size (1000,)



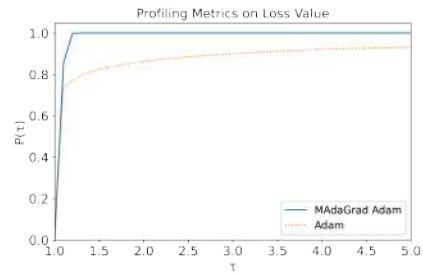
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)

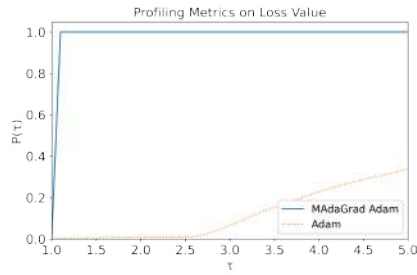


(e) Hidden layer size (100,100,100)

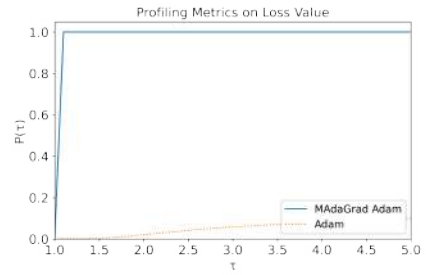


(f) Hidden layer size (100,100,100,100)

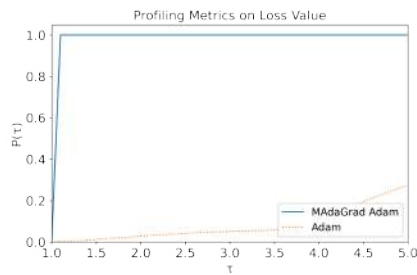
Figure 4.15: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Breast Dataset).



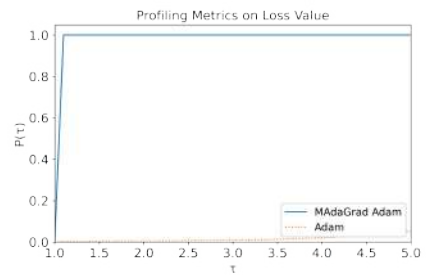
(a) Hidden layer size (100,)



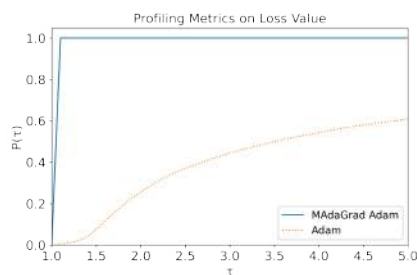
(b) Hidden layer size (1000,)



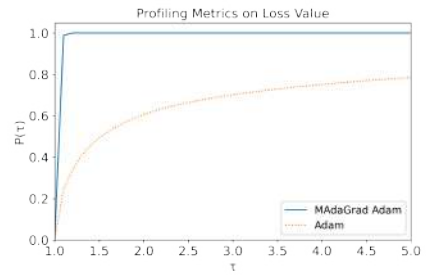
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)

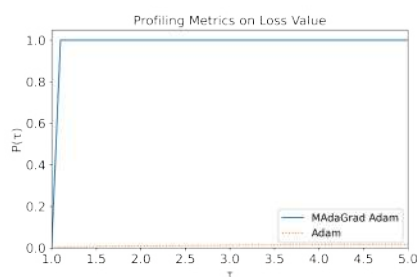


(f) Hidden layer size (100,100,100,100)

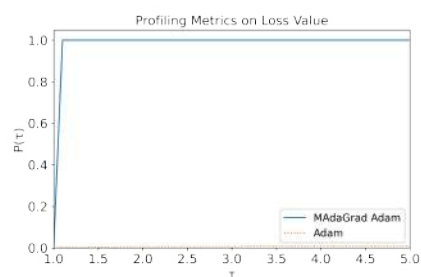
Figure 4.16: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Breast Dataset).

#### 4.4.4 Wine datasets

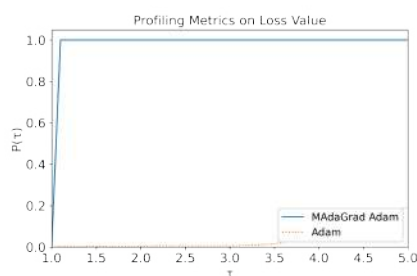
The performance for the wine datasets show the similar patterns compared to breast cancer datasets.



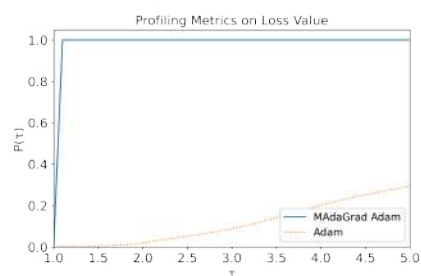
(a) Hidden layer size (100,)



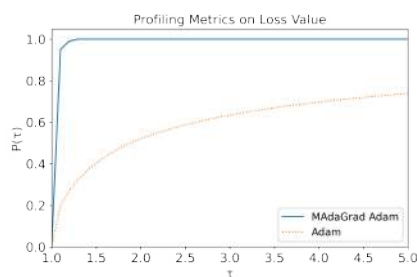
(b) Hidden layer size (1000,)



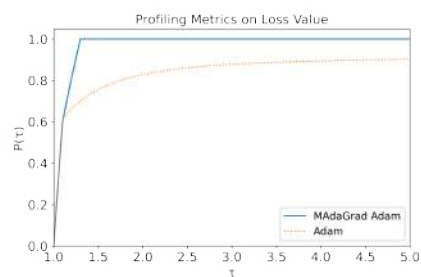
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)

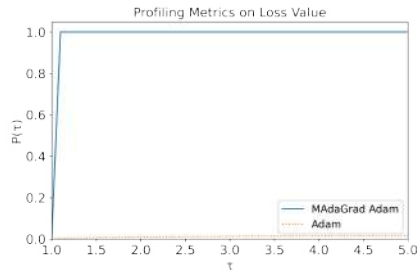


(f) Hidden layer size (100,100,100,100)

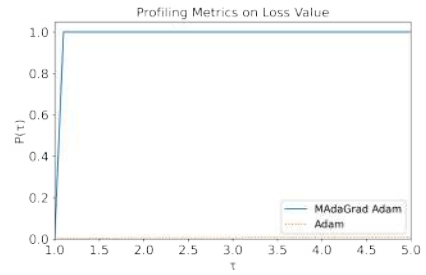
Figure 4.17: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Wine Dataset).

#### 4.4.5 Combined Datasets

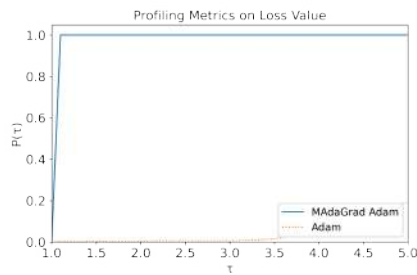
Figures 4.19 and 4.20 are the performance profiling graphs for the methods MAdaGrad Adam and Adam in terms of loss values for the batch sizes of 200 and 500. As the numbers of neurons and layers increase, the performance of Adam is improved. Although the difference in the value of  $P(\tau)$  between MAdaGrad Adam and Adam



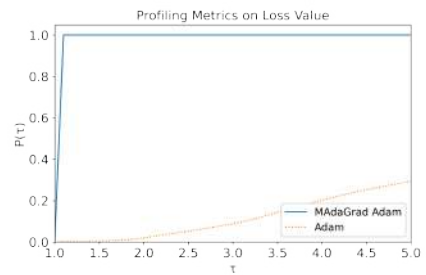
(a) Hidden layer size (100,)



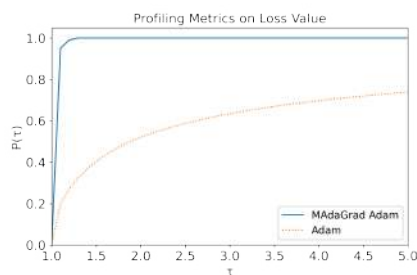
(b) Hidden layer size (1000,)



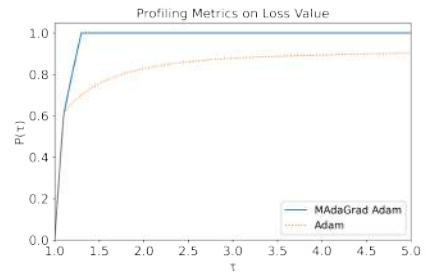
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



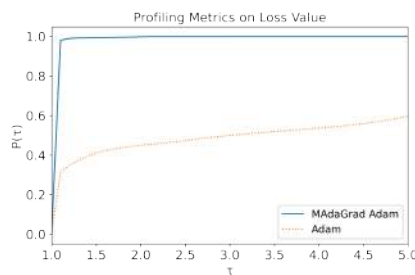
(e) Hidden layer size (100,100,100)



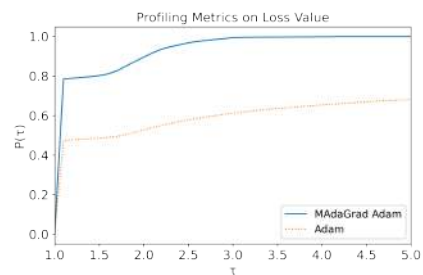
(f) Hidden layer size (100,100,100,100)

Figure 4.18: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Wine Dataset).

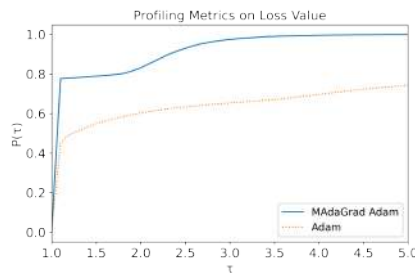
decreases, MAdaGrad Adam still performs better than Adam. The weight components in MAdaGrad Adam are optimized separately. As the number of neurons increases, more weight components are interconnected and show a little difference between optimization of MAdaGrad Adam and Adam. As the number of layers increases, it will lead to degradation of the performance of MAdaGrad Adam. Different batch sizes also give a similar pattern according to different hidden layer sizes.



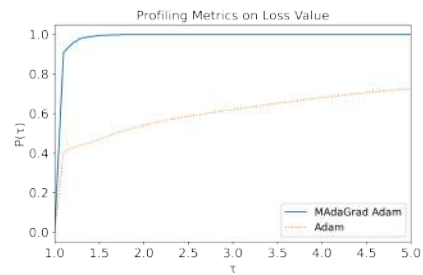
**(a) Hidden layer size (100,)**



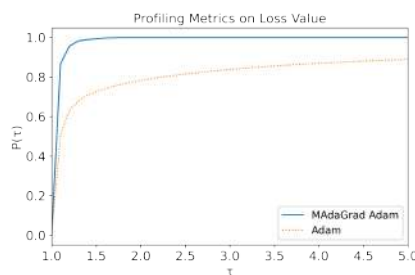
**(b) Hidden layer size (1000,)**



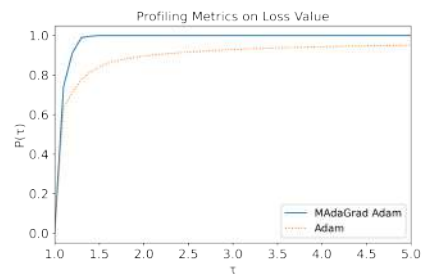
**(c) Hidden layer size (2000,)**



**(d) Hidden layer size (100,100)**

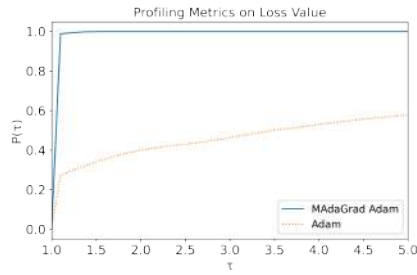


**(e) Hidden layer size (100,100,100)**

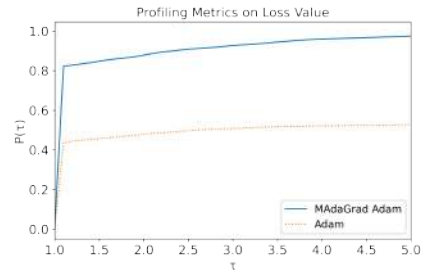


**(f) Hidden layer size (100,100,100,100)**

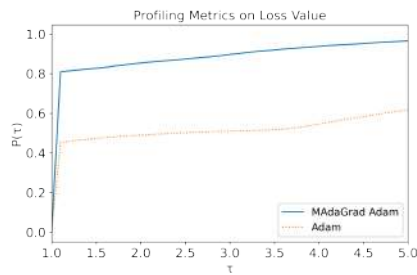
**Figure 4.19: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (Combined Dataset).**



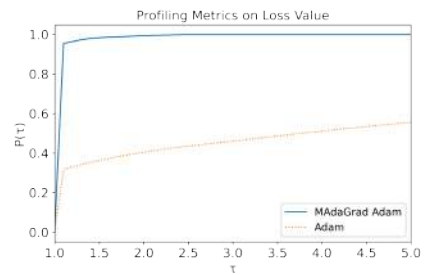
(a) Hidden layer size (100,)



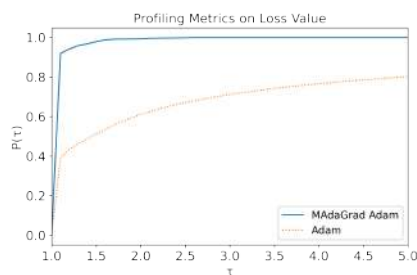
(b) Hidden layer size (1000,)



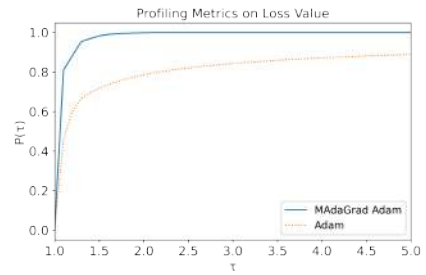
(c) Hidden layer size (2000,)



(d) Hidden layer size (100,100)



(e) Hidden layer size (100,100,100)



(f) Hidden layer size (100,100,100,100)

Figure 4.20: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 500 (Combined Dataset).

## 4.5 Comparison between MAdaGrad and SGD - Number of Iterations

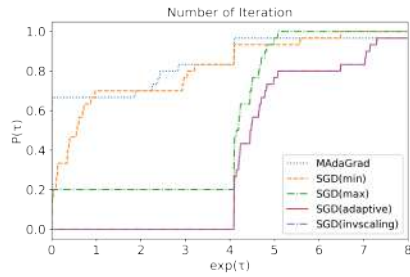
Figure 4.21 shows the performance profiling graphs of the methods, MAdaGrad, SGD(constant)(min), SGD(constant)(max), SGD(adaptive) and SGD(invscaling) for a batch size of 200. Based on the number of iterations required to converge, MAdaGrad gives the best performance among these methods for different hidden layer sizes. SGD(adaptive) and SGD(invscaling) produce the same results, both of them reach the maximum iterations. Besides, MAdaGrad performs better than SGD with minimum and maximum learning rates. From figure 4.21, SGD(min) gives comparable results to MAdaGrad for the hidden layer size (100,). As the number of neurons and number of hidden layers increase, MAdaGrad delivers better results than SGD(min). MAdaGrad produces outperforming results than SGD(max).

## 4.6 Comparison between MAdaGrad Adam and Adam - Number of Iterations

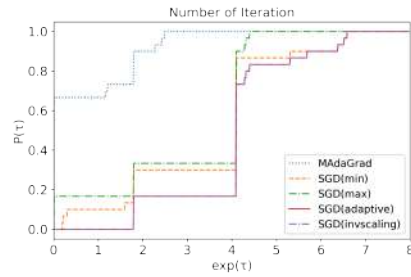
Figure 4.22 is the profiling graphs of MAdaGrad Adam and Adam in terms of number of iterations. In the hidden layer sizes (100,), (100,100) and (2000,), MAdaGrad Adam performs better than Adam in the first half, but not better than Adam in the second half. MAdaGrad Adam shows a better result than Adam for the hidden layer size (1000,) since the  $\mathbb{P}(t \leq \tau)$  directly increases to 1 at the beginning stage. For the hidden layer size (5000,), Adam indicates a better result than MAdaGrad Adam.

### 4.6.1 Loss values for MAdaGrad and SGD(constant)

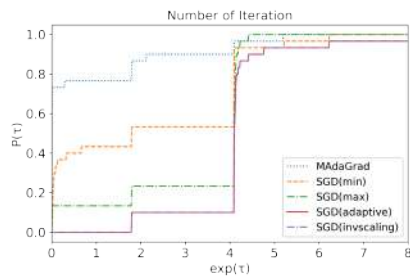
Figure 4.23 shows the loss values over 200 iterations for MAdaGrad and SGD(constant). In this training, the default batch size of 200 and the default hidden layer size (100,) are used. Figure 4.23 shows that the



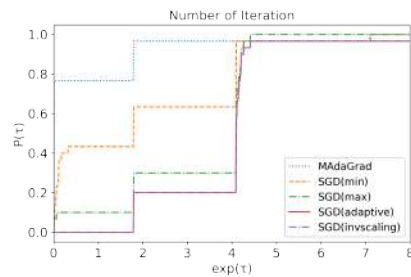
(a) Hidden layer size (100,)



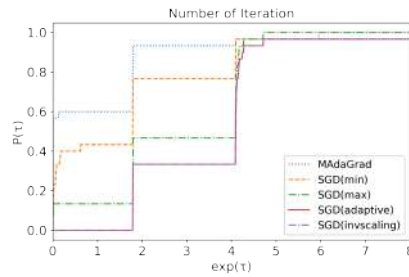
(b) Hidden layer size (100,100)



(c) Hidden layer size (1000,)



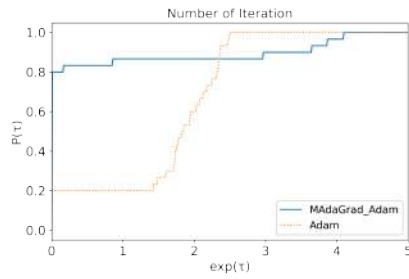
(d) Hidden layer size (2000,)



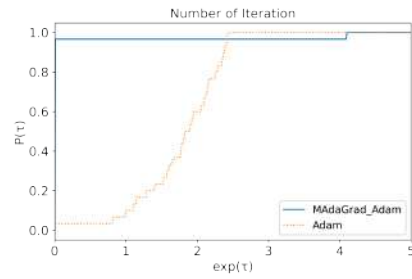
(e) Hidden layer size (5000,)

Figure 4.21: Profiling graphs of MAdaGrad and SGD for various layer sizes with a batch size of 200 (MNIST Dataset - Number of Iterations).

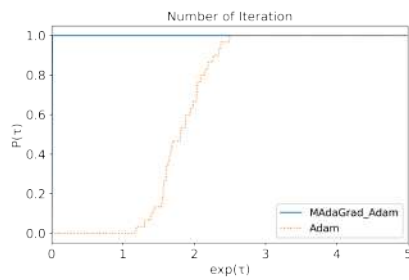




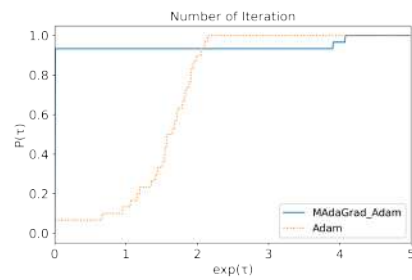
(a) Hidden layer size (100,)



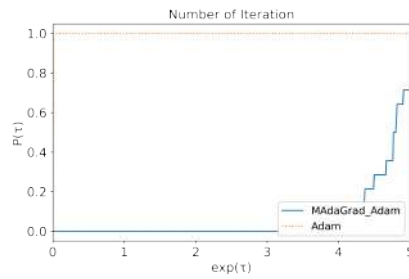
(b) Hidden layer size (100,100)



(c) Hidden layer size (1000,)



(d) Hidden layer size (2000,)



(e) Hidden layer size (5000,)

Figure 4.22: Profiling graphs of MAdaGrad Adam and Adam for various layer sizes with a batch size of 200 (MNIST Dataset - Number of Iterations).

MAdaGrad method performs better than SGD(constant). MAdaGrad performs better since the learning rate in MAdaGrad adapts for every component, instead of a fixed learning rate used throughout the training. SGD(Constant) and SGD(Adaptive) exhibit a similar pattern and they are overlapping, thus SGD(Adaptive) is not included in the figure. They show a similar pattern since they are the same if there are no two consecutive epochs that fail to decrease the learning rate by  $tol$  as mentioned in section 3.2.2.

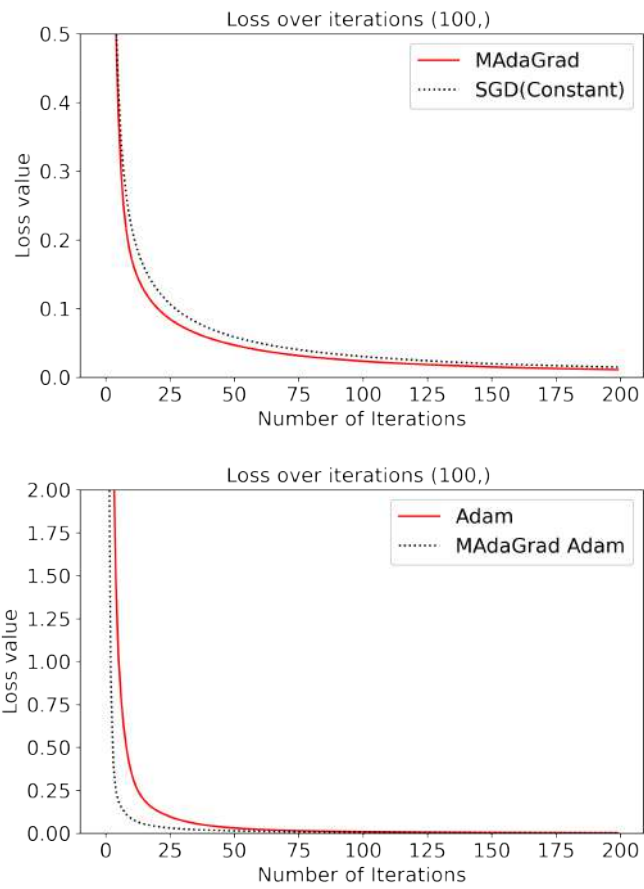
#### 4.6.2 Loss values for MAdaGrad Adam and Adam

Figure 4.23 shows the loss values over 200 iterations for MAdaGrad Adam and Adam. The default batch size of 200 and the default hidden layer size (100,) are used. Although MAdaGrad Adam does not show better performance than Adam according to profiling graphs in term of number of iterations, MAdaGrad Adam still performs better and converges faster at the beginning stage of the training.

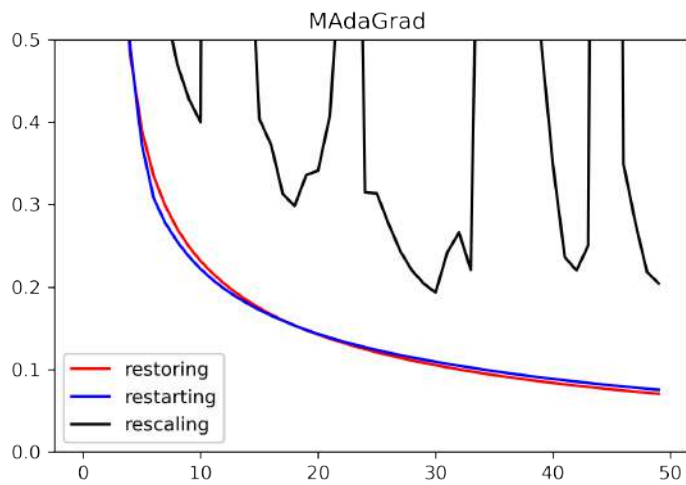
#### 4.7 Restoring, Restarting and Rescaling

Three types of choices are used for updating the cases  $s_k^T s_k \leq s_k^T y_k$  in 3.12. Figure 4.24 shows the loss value of the choices restoring, restarting and rescaling for MNIST digits datasets. Figure 4.24 shows that the rescaling method performs the worst among these methods. Restarting method shows a similar pattern but poorer performance than restoring method. The restarting method set  $B$  to identity matrix  $I$ , which goes back to the initial step, thus it does not give the best result. The restoring method performs the best among these methods, hence, restoring method is used in the updating formula of  $B_k$ .

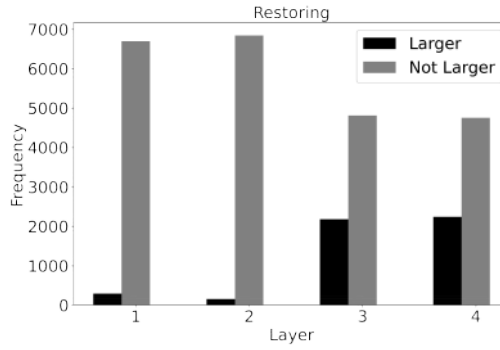
Figure 4.25a and 4.25b is the counts of the cases  $s_k^T s_k > s_k^T y_k$  and  $s_k^T s_k \leq s_k^T y_k$  in restoring and rescaling methods. The rescaling method is not



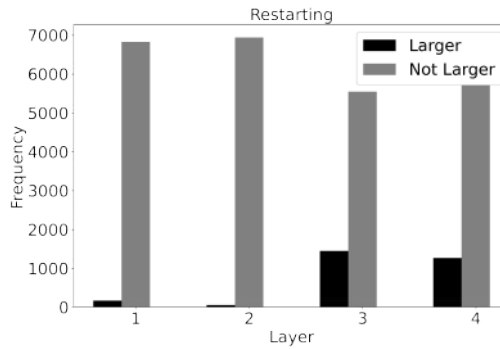
**Figure 4.23: Loss over iterations with hidden layer size (100,). (Top) MAdaGrad and SGD(constant). (Bottom) MAdaGrad Adam and Adam**



**Figure 4.24: Restoring, Restarting and Rescaling**



(a) Count in Restoring



(b) Count in Restarting

**Figure 4.25: Count in Restoring and Restarting.**

involved in the histograms, since it gives the worst performance among these three methods. The value counts for a neural network with 4 layers separately. Most of the time, the learning rate follows the branch  $s_k^T s_k \leq s_k^T y_k$  among the restoring and restarting methods, therefore, the diagonal part in the updating formula of the MAdaGrad method is rarely used.

## CHAPTER 5

### CONCLUSION AND FUTURE WORKS

#### 5.1 Conclusion

Optimization problems are a significant part of the neural network training process. One of the most important aspects of optimization is the learning rate. Finding optimal values for different hyperparameters yields training neural networks challenging. Nowadays, the common learning rate algorithms are either using a fixed learning rate in the whole simulation or adapting the learning rate in every iteration. In this project, a modified stochastic gradient descent (MAdaGrad) method and a modified adaptive moment estimation (MAdaGrad Adam) method are proposed. The proposed methods update the learning rate in every iteration based on the approximated spectrum of the Hessian of the loss function. Multiple adaptive learning rates are used for every single component. The MNIST handwritten digits datasets, Breast Cancer datasets, Wine datasets, and Abalone datasets are used to illustrate the performance of the proposed algorithms. There are two termination criteria used to observe the performance of the method, which are loss values and the number of iterations. Different hidden layer sizes are used to profile the performance of the proposed algorithms. The proposed algorithms are compared with the common existing methods SGD and Adam in terms of loss values. The proposed algorithm MAdaGrad is compared to the SGD method with the existing adaptive

learning rate schedule, invscaling learning rate schedule, and the constant learning rate schedule in scikit-learn. Besides, the proposed MAdaGrad Adam algorithm is compared with the well-known method Adam. Different batch sizes of 200 and 500 give a similar pattern according to different hidden layer sizes. The numerical results show that the proposed methods are comparable with SGD and Adam. To illustrate the performance of the proposed methods, the loss values over 200 iterations with the default batch size of 200 and hidden layer size (100,) are plotted. MAdaGrad performs better than SGD(constant) and SGD(invscaling) since the learning rate in MAdaGrad adapts for every component, instead of a fixed learning rate used throughout the training. MAdaGrad Adam performs better and converges faster than Adam at the beginning stage of the training. However, Adam performs slightly better than MAdaGrad Adam when it reached iteration 200, the loss value of Adam is lower than MAdaGrad Adam. Therefore, MAdaGrad and MAdaGrad Adam can be alternatives optimizer in machine learning.

## 5.2 Future Work

The selection of the learning rate will affect the performance of the methods. Instead of using 0.001 for the initial value of the learning rate  $\mu$ , future work may consider using different initial values to improve the efficiency of the methods. In the specific scenario that the Adam performs better than the MAdaGrad Adam method, a more appropriate initial value will deliver better performance of the proposed method. In addition, it is suggested to incorporate the nonmonotone line search strategy into the proposed method. Non-monotone line search techniques are essential components in optimization. It works with some combined conditions and provides the possibility to achieve better

performance. The non-monotone line search methods offer flexibility and adaptability in dealing with complex objective functions. The strategy assists in the exploration of a larger variety of solutions, the avoidance of local minima, and the efficient convergence of optimization algorithms in the presence of noise, ill-conditioning, and non-convexity. The proposed method combined with the non-monotone line search strategy will accelerate the convergence rate. It may speed up the proposed algorithm towards the optimum, especially when the initial step size estimates might be conservative.

## BIBLIOGRAPHY

Andrei, N. (2018), 'A diagonal quasi-newton updating method based on minimizing the measure function of byrd and nopedal for unconstrained optimization', *Optimization* **67**(9), 1553–1568.

Andrei, N. (2019), 'A new diagonal quasi-newton updating method with scaled forward finite differences directional derivative for unconstrained optimization', *Numerical Functional Analysis and Optimization* **40**(13), 1467–1488.

Andrei, N. (2021), 'A new accelerated diagonal quasi-newton updating method with scaled forward finite differences directional derivative for unconstrained optimization', *Optimization* **70**(2), 345–360.

Bae, K., Ryu, H. and Shin, H. (2019), 'Does adam optimizer keep close to the optimal point?', *arXiv preprint arXiv:1911.00289*.

Barzilai, J. and Borwein, J. (1988), 'Ima journal of numerical analysis', *Two-Point Step Size Gradient Methods* **8**, 141–148.

Becker, S. and Le Cun, Y. (1988), 'Improving the convergence of back-propagation learning with'.

Berahas, A. S., Curtis, F. E. and Zhou, B. (2022), 'Limited-memory bfgs with displacement aggregation', *Mathematical Programming* **194**(1), 121–157.

Bhargava, A. and Bansal, A. (2021), 'Novel coronavirus (covid-19) diagnosis using computer vision and artificial intelligence techniques: a review', *Multimedia tools and applications* **80**(13), 19931–19946.

Biglari, F. and Solimanpur, M. (2013), 'Scaling on the spectral gradient method', *Journal of Optimization Theory and Applications* **158**(2), 626–635.

Bock, S. and Weiß, M. (2019), A proof of local convergence for the adam optimizer, in '2019 International Joint Conference on Neural Networks (IJCNN)', IEEE, pp. 1–8.

Cartis, C., Gould, N. I. and Toint, P. L. (2010), 'On the complexity of steepest descent, newton's and regularized newton's methods for nonconvex unconstrained optimization problems', *Siam journal on optimization* **20**(6), 2833–2852.



- Cauchy, A. et al. (1847), 'Méthode générale pour la résolution des systèmes d'équations simultanées', *Comp. Rend. Sci. Paris* **25**(1847), 536–538.
- Chandra, B. and Sharma, R. K. (2016), 'Deep learning with adaptive learning rate using laplacian score', *Expert Systems with Applications* **63**, 1–7.
- Chen, Q., Leaman, R., Allot, A., Luo, L., Wei, C.-H., Yan, S. and Lu, Z. (2021), 'Artificial intelligence in action: addressing the covid-19 pandemic with natural language processing', *Annual review of biomedical data science* **4**, 313–339.
- Chen, X., Liu, S., Sun, R. and Hong, M. (2018), 'On the convergence of a class of adam-type algorithms for non-convex optimization', *arXiv preprint arXiv:1808.02941*.
- Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K. et al. (2012), 'Large scale distributed deep networks', *Advances in neural information processing systems* **25**.
- Dolan, E. D. and Moré, J. J. (2002), 'Benchmarking optimization software with performance profiles', *Mathematical programming* **91**(2), 201–213.
- Duchi, J., Hazan, E. and Singer, Y. (2011), 'Adaptive subgradient methods for online learning and stochastic optimization.', *Journal of machine learning research* **12**(7).
- Ge, R., Kakade, S. M., Kidambi, R. and Netrapalli, P. (2019), 'The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares', *Advances in Neural Information Processing Systems* **32**.
- Ghadimi, E., Feyzmahdavian, H. R. and Johansson, M. (2015), Global convergence of the heavy-ball method for convex optimization, in '2015 European control conference (ECC)', IEEE, pp. 310–315.
- Ghadimi, S. and Lan, G. (2013), 'Stochastic first-and zeroth-order methods for nonconvex stochastic programming', *SIAM Journal on Optimization* **23**(4), 2341–2368.
- Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S. and Lew, M. S. (2016), 'Deep learning for visual understanding: A review', *Neurocomputing* **187**, 27–48.
- Hestenes, M. R. and Stiefel, E. (1952), 'Methods of conjugate gradients for solving', *Journal of research of the National Bureau of Standards* **49**(6), 409.
- Johnson, R. and Zhang, T. (2013), 'Accelerating stochastic gradient descent using predictive variance reduction', *Advances in neural information processing systems* **26**.

- Kelley, C. T. (1995), *Iterative methods for linear and nonlinear equations*, SIAM.
- Kingma, D. P. and Ba, J. (2014), ‘Adam: A method for stochastic optimization’, *arXiv preprint arXiv:1412.6980* .
- Kochenderfer, M. J. and Wheeler, T. A. (2019), *Algorithms for optimization*, Mit Press.
- Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012), ‘Imagenet classification with deep convolutional neural networks’, *Advances in neural information processing systems* **25**.
- Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B. and Ng, A. Y. (2011), On optimization methods for deep learning, in ‘ICML’.
- LeCun, Y., Bengio, Y. and Hinton, G. (2015), ‘Deep learning’, *nature* **521**(7553), 436–444.
- Lee, D. and Yoon, S. N. (2021), ‘Application of artificial intelligence-based technologies in the healthcare industry: Opportunities and challenges’, *International Journal of Environmental Research and Public Health* **18**(1), 271.
- Li, Z. and Arora, S. (2019), ‘An exponential learning rate schedule for deep learning’, *arXiv preprint arXiv:1910.07454* .
- Liu, D. C. and Nocedal, J. (1989), ‘On the limited memory bfgs method for large scale optimization’, *Mathematical programming* **45**(1), 503–528.
- Liu, Q., Beller, S., Lei, W., Peter, D. and Tromp, J. (2022), ‘Pre-conditioned bfgs-based uncertainty quantification in elastic full-waveform inversion’, *Geophysical Journal International* **228**(2), 796–815.
- Liu, Y., Gao, Y. and Yin, W. (2020), ‘An improved analysis of stochastic gradient descent with momentum’, *Advances in Neural Information Processing Systems* **33**, 18261–18271.
- Loshchilov, I. and Hutter, F. (2016), ‘Sgdr: Stochastic gradient descent with warm restarts’, *arXiv preprint arXiv:1608.03983* .
- Luo, L., Xiong, Y., Liu, Y. and Sun, X. (2019), ‘Adaptive gradient methods with dynamic bound of learning rate’, *arXiv preprint arXiv:1902.09843* .
- Lydia, A. and Francis, S. (2019), ‘Adagrad—an optimizer for stochastic gradient descent’, *Int. J. Inf. Comput. Sci* **6**(5), 566–568.
- Montavon, G., Orr, G. and Müller, K.-R. (2012), *Neural networks: tricks of the trade*, Vol. 7700, springer.
- Morales, J. L. (2002), ‘A numerical study of limited memory bfgs methods’, *Applied Mathematics Letters* **15**(4), 481–487.
- Nesterov, Y. (2003), *Introductory lectures on convex optimization: A basic course*, Vol. 87, Springer Science & Business Media.

- Nesterov, Y. E. (1983), A method for solving the convex programming problem with convergence rate  $o(1/k^2)$ , in 'Dokl. akad. nauk Sssr', Vol. 269, pp. 543–547.
- Nocedal, J. (1980), 'Updating quasi-newton matrices with limited storage', *Mathematics of computation* **35**(151), 773–782.
- Park, J., Yi, D. and Ji, S. (2020), 'A novel learning rate schedule in optimization for neural networks and its convergence', *Symmetry* **12**(4), 660.
- Pennington, J., Socher, R. and Manning, C. D. (2014), Glove: Global vectors for word representation, in 'Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)', pp. 1532–1543.
- Polyak, B. T. (1964), 'Some methods of speeding up the convergence of iteration methods', *Ussr computational mathematics and mathematical physics* **4**(5), 1–17.
- Raydan, M. (1993), 'On the barzilai and borwein choice of steplength for the gradient method', *IMA Journal of Numerical Analysis* **13**(3), 321–326.
- Raydan, M. (1997), 'The barzilai and borwein gradient method for the large scale unconstrained minimization problem', *SIAM Journal on Optimization* **7**(1), 26–33.
- Reddi, S. J., Kale, S. and Kumar, S. (2019), 'On the convergence of adam and beyond', *arXiv preprint arXiv:1904.09237*.
- Robbins, H. and Monro, S. (1951), 'A stochastic approximation method', *The annals of mathematical statistics* pp. 400–407.
- Roux, N., Schmidt, M. and Bach, F. (2012), 'A stochastic gradient method with an exponential convergence rate for finite training sets', *Advances in neural information processing systems* **25**.
- Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747*.
- Schaul, T., Zhang, S. and LeCun, Y. (2013), No more pesky learning rates, in 'International conference on machine learning', PMLR, pp. 343–351.
- Sim, H. S., Leong, W. J. and Chen, C. Y. (2019), 'Gradient method with multiple damping for large-scale unconstrained optimization', *Optimization Letters* **13**(3), 617–632.
- Smith, L. N. (2015), 'No more pesky learning rate guessing games', *CoRR, abs/1506.01186* **5**, 363.
- Smith, L. N. (2017), Cyclical learning rates for training neural networks, in '2017 IEEE winter conference on applications of computer vision (WACV)', IEEE, pp. 464–472.

- Smith, L. N. (2018), 'A disciplined approach to neural network hyperparameters: Part 1—learning rate, batch size, momentum, and weight decay', *arXiv preprint arXiv:1803.09820* .
- Tato, A. and Nkambou, R. (2018), 'Improving adam optimizer'.
- Tieleman, T., Hinton, G. et al. (2012), 'Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude', *COURSERA: Neural networks for machine learning* **4**(2), 26–31.
- Tran, P. T. et al. (2019), 'On the convergence proof of amsgrad and a new version', *IEEE Access* **7**, 61706–61716.
- Xiao, Y., Wang, Q. and Wang, D. (2010), 'Notes on the dai–yuan–yuan modified spectral gradient method', *Journal of computational and applied mathematics* **234**(10), 2986–2992.
- Xiao, Y., Wei, Z. and Wang, Z. (2008), 'A limited memory bfgs-type method for large-scale unconstrained optimization', *Computers & Mathematics with Applications* **56**(4), 1001–1009.
- Xu, Z., Dai, A. M., Kemp, J. and Metz, L. (2019), 'Learning an adaptive learning rate schedule', *arXiv preprint arXiv:1909.09712* .
- Yi, D., Ahn, J. and Ji, S. (2020), 'An effective optimization method for machine learning based on adam', *Applied Sciences* **10**(3), 1073.
- Zeiler, M. D. (2012), 'Adadelta: an adaptive learning rate method', *arXiv preprint arXiv:1212.5701* .
- Zhang, M., Zhou, Y., Quan, W., Zhu, J., Zheng, R. and Wu, Q. (2020), 'Online learning for iot optimization: A frank–wolfe adam-based algorithm', *IEEE Internet of Things Journal* **7**(9), 8228–8237.
- Zhang, Z. (2018), Improved adam optimizer for deep neural networks, in '2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)', IEEE, pp. 1–2.
- Zhou, Z., Zhang, Q., Lu, G., Wang, H., Zhang, W. and Yu, Y. (2018), 'Adashift: Decorrelation and convergence of adaptive learning rate methods', *arXiv preprint arXiv:1810.00143* .
- Zinkevich, M. (2003), Online convex programming and generalized infinitesimal gradient ascent, in 'Proceedings of the 20th international conference on machine learning (icml-03)', pp. 928–936.

+