静岡大学 令和4年 博士論文

Stochastic Quasi-Newton Methods for Training Neural Networks

確率的準ニュートン法の高速化によるニューラルネットワークの高効率学習

指導教官　西村　雅史　教授

静岡大学・創造科学技術大学院自然科学系・情報科学専攻

提出　2022 年 6 月

氏名　Indrapriyadarsini Sendilkkumaar

学生番号　55945032

Stochastic Quasi-Newton Methods for Training Neural Networks

A thesis submitted in partial fulfillment for the degree of
Doctor of Philosophy

Indrapriyadarsini Sendilkkumaar
ID 55945032
Graduate School of Science and Technology
Shizuoka University

June 2022

Abstract

This thesis focuses on the study of stochastic quasi-Newton methods and ways to improve them for training neural networks using the Nesterov's acceleration scheme.

Neural networks have shown to be effective in a myriad of real-world applications. In recent years, stochastic first-order methods such as the stochastic gradient descent and its variants have taken the stage as the primary workhorse for training neural network models. This is mainly due to its affordable per-iteration computational costs which are linear. Several studies have been devoted to improving the convergence rates of first-order methods by introducing elegant ideas and algorithms, including acceleration techniques (E.g. Momentum, NAG), adaptive regularization (E.g. Adagrad, RMSprop), variance reduction (E.g. SAGA, SVRG), and many more.

On the contrary, second-order methods such as the Newton's method have typically been much less explored in training neural networks for large-scale problems, due to their prohibitive high per-iteration computational cost required for the computation of the Hessian and its inverse. Although Newton's method comes with good theoretical guarantees, these operations are infeasible for large-scale problems in high dimensions. Thus, a class of second-order quasi-Newton methods comes to the rescue, by offering iterative approximations to the computation of the inverse Hessian at a lower per-iteration cost. However, getting quasi-Newton methods to work in stochastic settings is a challenging task. Thus, this thesis aims at devising robust and efficient accelerated stochastic quasi-Newton methods for training neural networks. We propose a family of novel stochastic accelerated quasi-Newton methods, for non-convex optimization that attains fast convergence while maintaining a linear per-iteration cost, that match and improves over the performance of the best-known gradient-based methods.

The main contributions of this thesis are three-fold: First, we attempt to accelerate the conventional quasi-Newton methods with momentum and Nesterov's acceleration, with suitable modifications for large-scale stochastic optimization, where the objective function and gradients are noisy estimates computed from subsamples of the training data. We introduce direction normalization to reduce the stochastic variance. The algorithms are proposed in both its full and limited memory forms. We also provide discussions on the convergence rate and computational cost. The contents of this work are discussed in Chapter 4 of the thesis.

Next, we focus on devising a new stochastic Nesterov's accelerated quasi-Newton method suitable for training long-sequences modeled by recurrent neural networks (RNNs). The proposed algorithm targets to resolve the vanishing and/or exploding gradient issue that is common to RNNs by introducing direction normalization and control heuristics, along with acceleration using the Nesterov's gradient. We then extend the proposed accelerated stochastic algorithm for application to a deep reinforcement learning framework designed for solving an electronic design automation problem. The adaptability

of the proposed algorithm to both supervised and reinforcement learning frameworks confirms its robustness. The contents of this work are discussed in Chapter 5 and Chapter 6.

Finally, we investigate the feasibility of applying Nesterov's acceleration to other quasi-Newton methods, for both deterministic (full batch) and stochastic cases. We thus propose another Nesterov's accelerated quasi-Newton method that accelerates the symmetric rank-1 quasi-Newton method. The proposed method uses the trust-region approach and the convergence guarantee is provided. The results show that the performance of the proposed symmetric rank-1 method is significantly improved compared to the conventional symmetric rank-1 method. The contents of this work are discussed in Chapter 7 of the thesis.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Masafumi Nishimura for his support towards completing my dissertation. I would also like to extend my heartfelt gratitude to Prof. Hiroshi Ninomiya, Shonan Institute of Technology, for his valuable guidance and feedback towards my research. I would also like to thank Prof. Takeshi Kamio, Hiroshima City University, and my former supervisor Prof. Hideki Asai for their valuable support and engagement in my research activities. I also thank Prof. Atsuhiko Kai, Prof. Jun Kondoh and Prof. Masakatsu Nishigaki, for serving as my thesis review committee members along with Prof. Masafumi Nishimura and Prof. Hiroshi Ninomiya.

I thank all my labmates for their help and support both in lab and daily life. They were always friendly and helpful and maintained a cheerful environment that was conducive to learning. I would also like to thank Shahrzad Mahboubi from Shonan Institute of Technology. She has been a good friend and an active supporter in all my activities.

I am also extremely grateful to my undergraduate mentor, Mr. Sastry Ramachandrula, who ignited in me a spark towards research. He is an amazing mentor, well-wisher and friend, who always kept me motivated and grounded, over all these years.

I am grateful to everyone at Shizuoka University for providing a great environment for pursuing research. I owe immense gratitude to a lot of people who have made this journey memorable. Firstly, I would like to thank Prof. Shashidhar Tantry for introducing me to this opportunity to study abroad at Shizuoka University. Thanks to Prof. Mari Hakamata, Prof. Kanako Suzuki and all the language teachers for helping me learn Japanese, which not only helped with the daily life but also opened several opportunities during my stay in Japan. I am grateful to the Shizuoka University Asia Bridge Program, Shizuoka Bank Corporation and Honjo Foundation for the scholarships and funds that were generously awarded.

This journey would have never been pleasant without my amazing friends. Thanks to my roommate Alka Singh for being the person who I can always count on. I thank all my friends for always cheering and keeping me company, and building wonderful memories together.

I owe immense gratitute to Prof. Darius Greenidge for his valuable time and insights. I also thank the student counsellor Ms. Ishikawa and the international students counsellor Prof. Mari Hakamata for their support.

Last but not the least, I thank my family, parents and brother for motivating and supporting me in all walks of life.

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

Optimization forms the core in several areas of engineering, statistics, machine learning, neural networks, quantum computing, fundamental sciences, and more. In general, optimization is the minimization or maximization of a function, subject to constraints on its variables. A good optimization algorithm is expected to perform well across different types of problems (robustness) with reasonable computation and storage costs (efficiency) and less sensitivity to error and noise (accuracy) [1]. In the era of immense data, the effectiveness and efficiency of the optimization algorithms dramatically influence the popularization and application. The popularity of machine learning has been ever increasing over the last decade and has become one of the integral set of artificial intelligence and data science. Hence there is a dire need for solving large scale non-linear optimization problems.

Classical machine learning models are either convex or can be reduced to a convex optimization problems and can be efficiently solved using gradient based methods, ensuring convergence to a global optimum [2]. Much complex models especially those of deep learning (deep neural networks), however are difficult as they result in non-smooth, non-convex optimization problems.

Training deep neural networks poses several challenges such as ill-conditioning, high non-linearity of the objective function, hyperparameter tuning, overparmeterization etc. These challenges are usually addressed either by or in combination of model changes (such as increasing the depth of the network or number of hidden units), architectural changes (such as LSTMs and GRUs), changing the objective function (such as introducing regularization), and sophisticated weight update strategies or optimization techniques (such as Adam, Adagrad, adaHessian, etc). In this thesis, we focus on the optimization algorithms used in training neural networks. It is notable that optimization in deep neural networks (DNNs), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep reinforcement learning (DRL), each encounter different difficulties and challenges based on the problem considered.

For example, RNNs popularly used in NLP, are powerful sequence models. But despite their capabilities in modeling sequences, RNNs are particularly very difficult to train long sequences with long-term dependencies due to the vanishing and/or exploding gradient problem. Hence several algorithms and architectures have been proposed to address the issues involved in training RNNs.

Similarly, training neural networks in deep reinforcement learning tasks is usually slow and challenging, due to the training data being temporally correlated, non-stationary, and presented as a continuous stream of experiences rather than batches, which in contrast to supervised learning makes it more prone to unlearning effective features over time. Even in supervised learning, most real world

applications demand the model to be trained on a continuous stream of data (online), that are highly challenging to model due to the high stochastic noise and non-linearity and non-smoothness of the functions. Increasing the number of parameters or depth of the network has always been the first resort for solving high non-linearity. However studies on overparameterization [3, 4] suggest that increasing the depth of the neural network architecture leads to faster training as a result of increased expressive power, and not an outcome of the so-called robust optimizer itself. Moreover, overparameterization coupled with large scale optimization and the immense amount of data that needs to be processed in training a large neural network further increases the load on computation and storage costs. Hence, there is a trade-off between the scale of the network and its expressive power and thus a robust optimizer must not only perform well on large networks, but also on smaller neural networks.

1.1 Motivation and Objective

Machine learning models have made remarkable impact in several real-world applications. These machine learning models eventually reduce to an optimization problem that are solved with conventional mathematical optimization methods. Iterative gradient based algorithms have been widely used in optimization and have been actively researched in order to devise robust and efficient algorithms that result in accurate solutions. These algorithms can be broadly categorized as (1) first-order methods (eg. SGD, Adam) (2) higher order methods (eg. Newton method, quasi-Newton method) and (3) heuristic derivative-free methods (eg. coordinate descent, SPSA), each with its own pros and cons. Apart from introducing sophisticated update strategies, several works are also dedicated toward acceleration techniques (eg. momentum method, Nesterov's acceleration, Anderson's acceleration). Much progress has been made in the last 20 years in designing and implementing robust and efficient methods and yet there are many classes of applications where current state of the art optimizers fails.

Optimization in machine learning is presently dominated by first-order methods such as SGD and Adam. However, these methods come with well-known issues such as slow convergence, sensitivity to hyperparameter settings, stagnation at high training errors, and difficulty escaping flat regions and saddle points. These problems are significantly prominent and prone to in highly non-convex settings such as in the case of neural network training. On the other hand, second-order methods are among the most powerful algorithms in mathematical optimizations. Recent studies in second-order methods have shown to alleviate these shortcomings by capturing the curvature information. Despite the strong theoretical properties, second-order methods are less prevalent in machine learning and deep learning due to the high computation and storage costs. However, there have been several recent studies under quasi-Newton methods that show great efficiencies with convergence in fewer steps or iterations and at a moderate computational cost with better scalability for large scale optimization.

In light of the above, this thesis aims to investigate if acceleration techniques such as introducing momentum or Nesterov's acceleration to second-order methods outperform conventional methods and avoid overparameterization, and more importantly, if they are robust, efficient and practical. To this end, we study the efficiency, robustness and accuracy of the Nesterov's acceleration applied to quasi-Newton methods and develop practical algorithms for real-world problems. The main contributions of this thesis are three-fold. First we attempt to accelerate the conventional quasi-Newton methods with momentum and Nesterov's acceleration, with suitable modifications for large scale stochastic optimization. Next, we focus on devising a new stochastic Nesterov's accelerated quasi-Newton method suitable for training

long-sequence models and extending its application to deep reinforcement learning framework to confirm robustness across applications. Lastly, we investigate the feasibility of Nesterov's acceleration applied to other quasi-Newton methods such as the symmetric rank-1 method. Both theoretical and empirical analysis of the proposed algorithms are discussed. The main objectives of this thesis can be listed as follows:

- Study the fundamentals of first-order and second-order methods in training neural networks.
- Investigate the feasibility of Nesterov's acceleration on algorithms in the quasi-Newton family.
- Devise robust and efficient accelerated second-order optimizers suitable for stochastic training.
- Analyze computational cost and convergence guarantees.

1.2 Thesis Outline

The rest of the thesis is organized as follows:

- **Chapter 2** : This chapter introduces the basics of optimization in the context of training neural networks and gives an overview of common gradient based optimization algorithms used in neural networks.
- **Chapter 3** : This chapter gives an overview of quasi-Newton methods for training neural networks. The performance of quasi-Newton methods in comparison to first-order methods is studied in the deterministic (full batch) setting on a few function approximation problems and applications related to circuit modeling and design optimization.
- **Chapter 4** : This chapter discusses stochastic training of neural networks with quasi-Newton methods for large scale optimization. Stochastic extensions of the Nesterov and momentum accelerated BFGS methods are proposed along with their convergence analysis.
- **Chapter 5** : This chapter focuses on recurrent neural networks. The common problems in training recurrent neural networks are studied and an adaptive stochastic Nesterov's accelerated quasi-Newton method is proposed along with its convergence rate analysis.
- **Chapter 6** : This chapter investigates the utility of quasi-Newton optimization methods in deep reinforcement learning applications. In extension to the work in the previous chapter, the accelerated stochastic quasi-Newton method is adapted to training deep Q-networks. With the example of solving VLSI global routing using deep reinforcement learning, a combinatorial optmization problem, the efficiency and robustness of the proposed method is demonstrated.
- **Chapter 7** : This chapter investigates the feasibility of introducing Nesterov's acceleration to other members of the quasi-Newton family. This chapter introduces momentum and Nesterov's acceleration to the SR1 method, a low rank quasi-Newton method, in deterministic and stochastic training.
- **Chapter 8** : This chapter concludes with summarizing the contributions of this study along with future research directions.

Each chapter is self-contained and the necessary notations and backgrounds are introduced in each chapter.

1.3 Notation

We breifly describe in this section the notations followed throughout the thesis. Lowercase letters are used to denote scalars and vectors while uppercase letters are used to denote matrices. In general, all vectors are column vectors denoted by boldface lowercase characters (Eg. $\mathbf{w} \in \mathbb{R}^d$). The matrices are denoted by boldface uppercase characters (Eg. $\mathbf{X} \in \mathbb{R}^{d \times d}$) and scalars by simple lowercase characters (Eg. μ). The superscript $^{\mathrm{T}}$ denotes the transpose of the vector or matrix. Non-bold uppercase characters (such as X) are used to denote sets. The scalars, vectors and matrices at each iteration bear the corresponding iteration index k as a subscript. The "big O" notation $O(\cdot)$ is used for the computational complexity. We use the notation $\|\cdot\|$ to denote the L_2 or Euclidean norm i.e., $\|\mathbf{x}\| := \sqrt{\mathbf{x}^{\mathrm{T}}\mathbf{x}}$. In addition to the above, the following symbols and notation are reserved throughout the thesis.

- iteration index $k \in \mathbb{N} : k = 1, 2, ..., k_{max} \in \mathbb{N}$
- n is the number of total samples in T_r and is given by $|T_r|$.
- b is the number of samples in the minbatch $X \subset T_r$ and is given by $|X|$.
- d is the number of parameters of the neural network.
- m is the limited memory size.
- α_k is the learning rate or step size.
- μ_k is the momentum coefficient, chosen in the range (0,1).
- $E(\mathbf{w})$ is the error evaluated at $\mathbf{w}$.
- $\nabla E(\mathbf{w})$ is the gradient of the error function evaluated at $\mathbf{w}$.

2

Optimization for Training Neural Networks

2.1 Introduction

Many machine learning models are often cast as continuous optimization problems in multiple variables. In supervised machine learning, given a dataset T_r with n samples, each sample comprises of an input data and output label pair (i_p, d_p), and a parameterized model maps the function from the set of input data to the labels. These parameterized models could be decision trees, SVMs, neural networks, etc. In this thesis, we focus on neural network models.

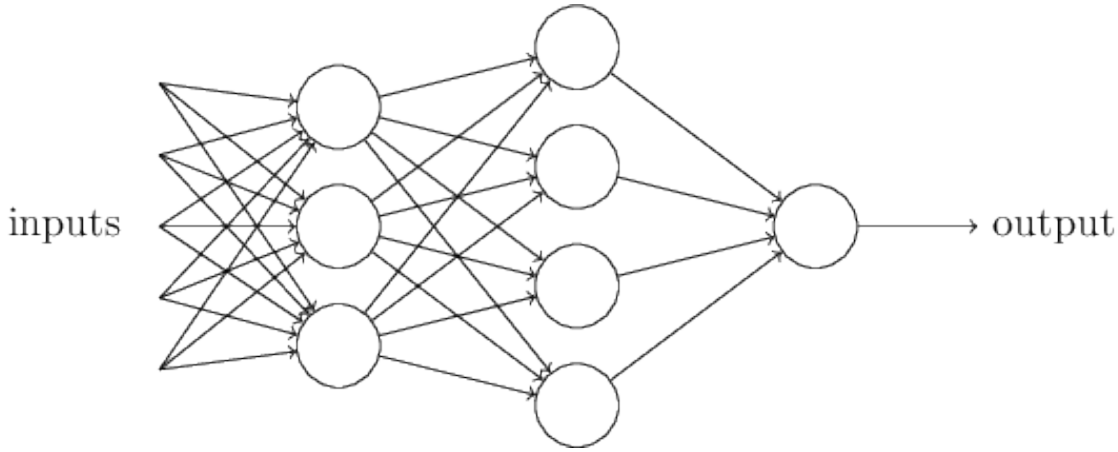


Figure 2.1: A simple neural network

Inspired by the concept of their biological neural network, artificial neural networks have gained immense popularity in the field of machine learning and artificial intelligence. Neural networks and deep learning currently provide the best solutions to many problems in image recognition, speech recognition, and natural language processing. A typical neural network consists of an input layer, hidden layer(s) and an output layer interconnected by parameters called weights. Each unit of a layer is called a neuron. Neural networks with a few hidden layers are called shallow neural networks while networks with several hidden layers are called deep neural networks. Neural networks have been developed from a simple architecture to a more and more complex structure, such as convolutional neural networks and recurrent neural networks, for practical applications.

Training a neural network involves updating the weights of the network in such a way that the error between the outputs predicted by the neural networks and the actual response being modeled is

minimized. This essentially comes down to an optimization problem where the objective is to minimize the total error, which in other words is an empirical risk minimization problem. Formally, the empirical risk is given as an error or loss function $E(\mathbf{w})$ where $\mathbf{w} \in \mathbb{R}^d$ are the parameters or weights of the neural network model under consideration. The error function $E(\mathbf{w})$ can be expressed as a sum of losses (risks) over individual training samples.

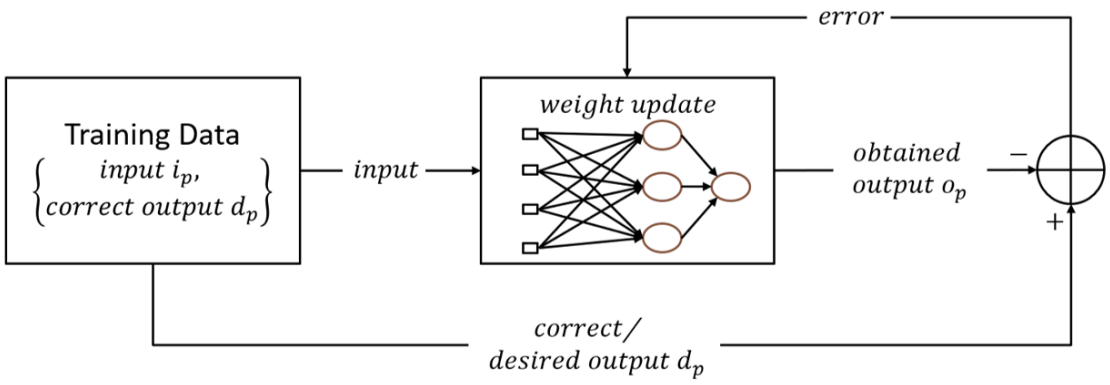


Figure 2.2: A simple block diagram of neural network training

A popular method of training neural networks is the error back-propagation (BP) algorithm which uses gradient descent to systematically modify or update the weights of the neural network by an amount proportional to the partial derivative of the accumulated error function, with respect to a given weight. The goal with backpropagation is to update each of the weights in the network so that they cause the neural network predicted output to be closer to the desired output (true label), by minimizing the error for each output neuron and the network as a whole.

2.2 Gradient Based Optimization

Machine learning algorithms often rely on optimization of some objective function, thus making the choice of the optimization algorithm a crucial part of the learning. Training in neural networks is an iterative process in which the parameters are updated in order to minimize an objective function. Given a dataset T_r with samples $(i_p, d_p)_{p \in T_r}$ drawn at random, and a function $E_p(\mathbf{w}; i_p, d_p)$ parameterized by a vector $\mathbf{w} \in \mathbb{R}^d$, the objective function is defined as

$$\min_{\mathbf{w} \in \mathbb{R}^d} E(\mathbf{w}) = \frac{1}{|T_r|} \sum_{p \in T_r} E_p(\mathbf{w}), \tag{2.1}$$

where $E_p(\mathbf{w})$ is the loss or error function. In case of regression problems, the most commonly used error function is the mean squared error (MSE), also known as the L2 loss, and is defined as,

$$E_p(\mathbf{w}) = \frac{1}{2}(d_p - o_p)^2 \tag{2.2}$$

where d_p and o_p are the desired output (true label) and neural network predicted output, respectively. For classification problems, cross entropy error, also known as the log loss is widely used. The cross entropy error for a multi-class classification problem with M classes is defined as follows.

$$E_p(\mathbf{w}) = -\sum_{c=1}^{M} d_p \log o_p. \tag{2.3}$$

As discussed earlier, in order to minimize the empirical risk given in (2.1), a natural approach is to use a gradient method. In gradient based methods, the objective function $E(\mathbf{w})$ under consideration is minimized by the iterative formula,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_{k+1}, \tag{2.4}$$

where k is the iteration count and $\mathbf{v}_{k+1}$ is the update vector, which is defined for each gradient algorithm. In case of the simple gradient descent algorithm, $\mathbf{v}_{k+1}$ is given as

$$\mathbf{v}_{k+1} = -\alpha_k \nabla E(\mathbf{w}_k), \tag{2.5}$$

where α_k is the learning rate or step size and $\nabla E(\mathbf{w}_k)$ is the gradient of the objective function computed with respect to $\mathbf{w}_k$ over all samples in the training set, i.e.,

$$\nabla E(\mathbf{w}_k) = \frac{1}{|T_r|} \sum_{p \in T_r} \nabla E_p(\mathbf{w}_k). \tag{2.6}$$

This type of training is known as full batch or deterministic training, where the algorithms used to train the neural networks require computation of the gradient with respect to the parameters $\mathbf{w}$ over all samples in the training set. In other words, the weights of the neural network are updated based on the expected sum of gradients computed over all samples in the training dataset. Typically the cost of gradient computation is of the order $O(nd)$ where n is the number of samples in the training set and d is the number of parameters of the neural network. It is thus evident that as the scale of the network increases and if the number of training examples is extremely large, it can result in huge computational and storage costs. Thus stochastic or mini-batch training approaches have become popular. In stochastic training, the objective function and gradients are computed over a smaller subset, $X \subseteq T_r$ of size b, randomly drawn from of the training data. Note that we denote the stochastic objective function with the subscript b as

$$E_b(\mathbf{w}) = \frac{1}{|X|} \sum_{p \in X} E_p(\mathbf{w}), \tag{2.7}$$

and the gradient $\nabla E_b(\mathbf{w}_k, X_k)$ denotes the gradient computed over the mini sample X_k.

$$\nabla E_b(\mathbf{w}_k, X_k) = \frac{1}{|X_k|} \sum_{p \in X_k} \nabla E_p(\mathbf{w}_k) \tag{2.8}$$

Stochastic training is faster compared to the full batch approach and the computation cost is reduced to $O(bd)$ as the function and gradient are evaluated on a smaller sub-sample with $b << n$. However, the stochastic gradients are noisy estimates of the full gradient. Therefore, each update step is noisy, resulting in more update steps being required for convergence. This gave rise to the need for several stochastic variance reduction techniques such as in [5–8].

Training neural networks can be challenging as the objective functions modeled by a neural network are usually non-convex and might have local optima, flat regions and saddle points in the loss surface [2]. As a result, the learning process might be too slow or arrive at a poor solution. The simplest approach to address both flat regions and differential curvature is to adjust the gradients in some way to account for poor convergence. This includes implicitly using the curvature to adjust the gradients of the objective function with respect to each parameter. Examples of such techniques include the pairing of vanilla gradient-descent methods with computational algorithms like the momentum method, NAG, RMSProp, or Adam. There are yet another class of methods that uses second-order derivatives to

explicitly measure the curvature. A second derivative provides the rate of change in gradient, which is a direct measure of the unpredictability of using a constant gradient direction over a finite step. The second-derivative matrix or the Hessian matrix thus contains useful information about the directions along which the greatest curvature occurs and is used by techniques like the Newton method in order to adjust the directions of movement by using a trade-off between the steepness of the descent and the curvature along a direction.

In the following sections, we briefly discuss on some of the first and second-order gradient based algorithms with a simple example. Consider an arbitrary function with two parameters $\{x_1, x_2\}$ defined as follows.

$$f(x) = \frac{1}{2}x_1^2 + \frac{5}{2}x_2^2 - x_1x_2 - 2(x_1 + x2) \tag{2.9}$$

with the global minimum located at $\{x_1, x_2\} = \{3, 1\}$. We set the initial points of $\{x_1, x_2\} = \{2, 1\}$, and visualize the trajectories of the algorithm on a contour plot.

2.3 First Order Gradient Based Optimization

In case of first-order methods, the weights of the neural network are updated using only the gradient information or the first-order derivative. Thus owing to the simplicity in implementation and low computational complexity, optimization in machine learning and neural network training have been dominated by first-order gradient methods. For large scale problems too, several works have been studied under stochastic first-order methods and its variance-reduced and accelerated variants [5, 9–17]. However, first-order methods are more sensitive to hyperparameter tuning and often exhibit slow convergence. Some of the popular first-order methods are discussed briefly below.

2.3.1 Steepest Gradient Descent

The gradient descent method is one of the earliest and simplest algorithm for finding the minimum of an objective function. The gradient descent method is based on the fact that the gradient of a function always points in the direction of maximum increase, and hence by moving in the direction opposite to that of the gradient will result in an improvement in minimizing the value of the objective function. Therefore, the update vector $\mathbf{v}_k$ is given as

$$\mathbf{v}_{k+1} = -\alpha_k \nabla E(\mathbf{w}_k) \tag{2.10}$$

The algorithm of the gradient descent method is given in Algorithm 2.1. The learning rate α_k is an important parameter in SGD as it determines the step size along the direction of the gradient $\nabla E(\mathbf{w}_k)$. If the step size is too large, the behaviour of the algorithms will get very noisy which can lead to inability to converge to a reasonable value. If the step size is too small however, the algorithm may get stuck in an area where the gradient is small. In practice it is common to let the learning rate be fixed, adaptive or use simple decay schedules such as,

$$\alpha_k = \frac{\tau}{\tau + k}\alpha_0 \tag{2.11}$$

There are also variants of SGD that uses line search to determine the optimal step size at each iteration. The gradient descent method although simple, has been successful in several problems. However, the

convergence can be slow and may even also not be suitable for complex and highly non-linear problems. Figure 2.3 below show the trajectory of the gradient descent algorithm for the function in (2.9). As seen from the figure, it took 169 steps to converge to the optimum point with a learning rate of 0.1. The learning rate was chosen from the set $\{0.001, 0.01, 0.1, 1\}$, that gave the least number of steps to converge.

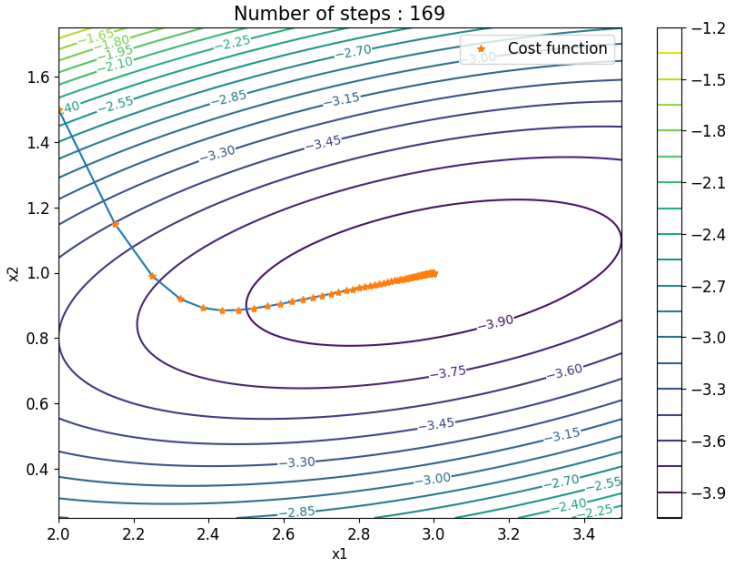


Figure 2.3: Trajectory of the gradient descent with step size $\alpha = 0.1$

2.3.2 Momentum Acceleration

Classical Momentum

The classical momentum method (CM) accelerates the gradient descent method by accumulating previous vector updates in direction of persistent reduction [18]. The momentum based methods are set to address issues with escaping local minima and flat regions. The momentum term in mathematics is built on the physics analogy of a heavy ball rolling downhill gains a momentum velocity along its trajectory, and hence this method is also known as the heavy ball method. The update vector of the

Algorithm 2.1 GD Method

Require: ε and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$.
1: $k \leftarrow 1$
2: **while** $\|E(\mathbf{w}_k)\| > \varepsilon$ and $k < k_{max}$ **do**
3: Calculate $\nabla E(\mathbf{w}_k)$
4: $\mathbf{v}_{k+1} \leftarrow -\alpha_k \nabla E(\mathbf{w}_k)$
5: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
6: $k \leftarrow k + 1$
7: **end while**

momentum method is given by:

$$\mathbf{v}_{k+1} = \mu \mathbf{v}_k - \alpha_k \nabla E(\mathbf{w}_k). \tag{2.12}$$

where $\mu \in (0, 1)$ denotes the momentum term. The momentum method is shown in Algorithm 2.2. Further, Figure 2.4 shows the trajectory of the momentum method for the function in (2.9) and it can be observed that with a momentum $\mu = 0.8$, the momentum method converges faster by 58 steps for the same learning rate of 0.1.

Algorithm 2.2 Momentum Method

Require: $0 < \mu_k < 1$, ε and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$ and $\mathbf{v}_k = 0$.
1: $k \leftarrow 1$
2: **while** $\|E(\mathbf{w}_k)\| > \varepsilon$ and $k < k_{max}$ **do**
3: Calculate $\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$
4: $\mathbf{v}_{k+1} \leftarrow \mu_k \mathbf{v}_k - \alpha_k \nabla E(\mathbf{w}_k)$
5: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
6: $k \leftarrow k + 1$
7: **end while**

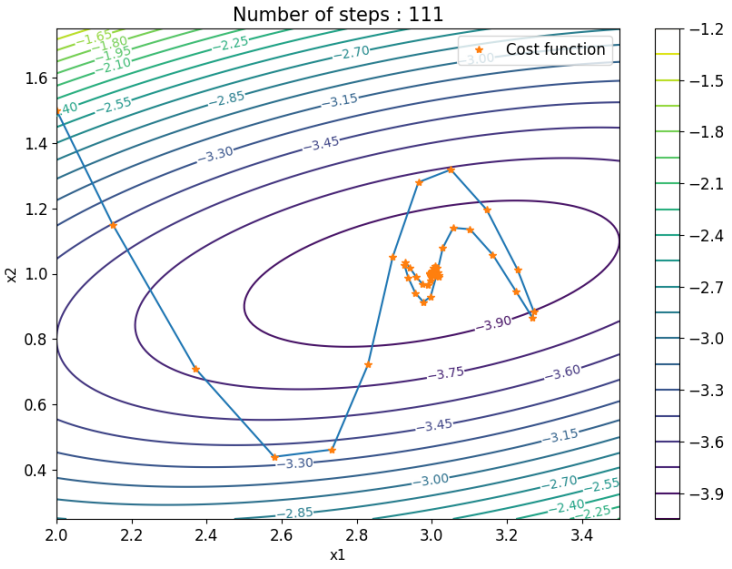


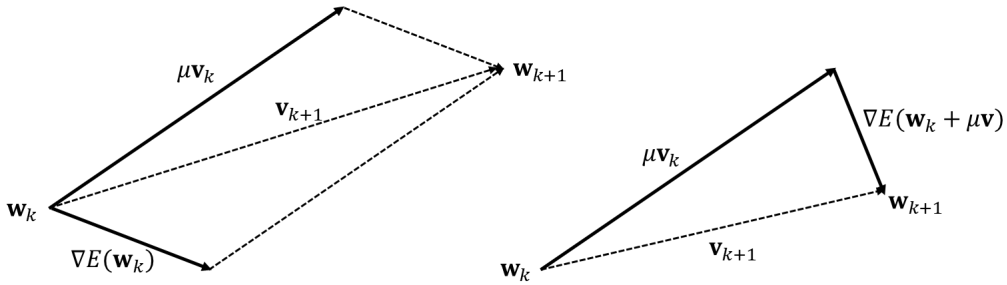
Figure 2.4: Trajectory of the classical momentum method with $\alpha = 0.1$ and $\mu = 0.8$

Nesterov's Accelerated Gradient

The Nesterov's Accelerated Gradient (NAG) method is a modification of the classical momentum method in which the gradient is computed at $\mathbf{w}_k + \mu \mathbf{v}_k$ instead of $\mathbf{w}_k$ [20]. Thus, the update vector is given by:

$$\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k - \alpha_k \nabla E(\mathbf{w}_k + \mu \mathbf{v}_k). \tag{2.13}$$

where $\nabla E(\mathbf{w}_k + \mu \mathbf{v}_k)$ is the gradient at $\mathbf{w}_k + \mu \mathbf{v}_k$ and is referred to as Nesterov's accelerated gradient vector. By computing the gradient at a point along the momentum direction and then taking a step update



(a) Classical momentum method (b) Nesterov's accelerated gradient method

Figure 2.5: Vector representation of CM and NAG update steps (Reproduced from [19])

results in further acceleration of the gradient descent method compared to the classical momentum method. The algorithm is as shown in Algorithm 2.3. Figure 2.5 shows the vector representation of the classical momentum and Nesterov's accelerated gradient method.

Algorithm 2.3 NAG Method

Require: $0 < \mu_k < 1$, ε and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$ and $\mathbf{v}_k = 0$.
1: $k \leftarrow 1$
2: **while** $\|E(\mathbf{w}_k)\| > \varepsilon$ and $k < k_{max}$ **do**
3: Calculate $\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$
4: $\mathbf{v}_{k+1} \leftarrow \mu_k \mathbf{v}_k - \alpha_k \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$
5: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
6: $k \leftarrow k + 1$
7: **end while**

From Figure 2.4 and Figure 2.6 we can observe that for the same learning rate $\alpha_k = 0.1$ and momentum factor $\mu = 0.8$, the Nesterov's accelerated gradient converges to the optimum in 79 steps while the classical momentum took 111 steps. However both the momentum accelerated methods converged in fewer steps compared to the vanilla gradient descent method.

2.3.3 Other Update Strategies

Apart from accelerating the gradient descent method using momentum or Nesterov's gradient, several methods have been proposed that devise sophisticated update strategies such as [12, 21–25] that have shown to perform better than the vanilla gradient descent method. Among these, the methods most commonly used in neural network training are briefly discussed below.

AdaGrad

Adagrad [12] is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, thus performing smaller updates. Adagrad uses a per coordinate step size which depends on the scale of past gradients. The update vector is given as

$$\mathbf{v}_{k+1} = -\frac{\alpha}{\sqrt{\sum_k (\nabla E(\mathbf{w}_k))^2 + \epsilon}} \nabla E(\mathbf{w}_k). \tag{2.14}$$

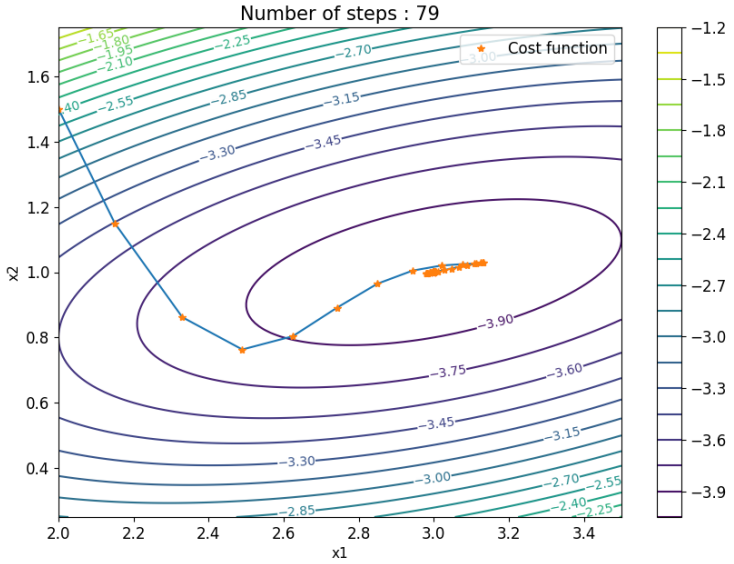


Figure 2.6: Trajectory of the Nesterov's accelerated gradient method with $\alpha = 0.1$ and $\mu = 0.8$

α is a global stepsize shared by all dimensions and is usually set to $\alpha = 0.01$. Adagrad proved effective for sparse optimization but when applied to deep learning, it was under-performing [26]. This was due to the large impact of past gradients which prevented it from adapting to local changes in the smoothness of the objective function.

RMSprop

The RMSprop method [21] improves upon the AdaGrad method by using an exponential moving average instead of a cummulative sum to reduce the impact of past gradients. The update vector is given by

$$\mathbf{v}_{k+1} = -\frac{\alpha}{\sqrt{\theta_{k,i} + \epsilon}} \nabla E(\mathbf{w}_k), \tag{2.15}$$

where

$$\theta_k = \gamma \theta_{k-1} + (1 - \gamma)(\nabla E(\mathbf{w}_k))^2. \tag{2.16}$$

where $\epsilon = 10^{-8}$ and θ_k is the exponential moving average of the gradient. γ is the decay factor and α is the global stepsize, usually set to 0.9 and 0.001 respectively.

Adam

Adam is one of the most popular and effective first-order methods and is based on adaptive estimates of the first and second moments. The update rule in the algorithm is based on exponentially decaying moving average of past squared gradients and exponentially decaying moving average of past gradients [22]. The algorithm introduces two new hyperparameters $0 \leq \beta_1, \beta_2 < 1$ that control the exponential decay rates of these running averages. The running average themselves are estimates of the first moment (the mean) and the second raw moment (the uncentered variance) of the gradient. β_1 and β_2 are chosen

to be 0.9 and 0.999, respectively. The estimates $\mathbf{m}$ and θ when initialized to zero get biased towards zero and are corrected by the bias-corrected estimates $\hat{\mathbf{m}}$ and $\hat{\theta}$.

$$\mathbf{v}_{k+1} = -\alpha \frac{\hat{\mathbf{m}}_k}{(\sqrt{\hat{\theta}_k} + \epsilon)}, \tag{2.17}$$

where

$$\hat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{(1 - \beta_1^k)}, \tag{2.18}$$

$$\hat{\theta}_k = \frac{\theta_k}{(1 - \beta_2^k)}, \tag{2.19}$$

where, $\mathbf{m}_k$ and θ_k given by:

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1 - \beta_1)\nabla E(\mathbf{w}_k), \tag{2.20}$$

$$\theta_k = \beta_2 \theta_{k-1} + (1 - \beta_2)(\nabla E(\mathbf{w}_k))^2. \tag{2.21}$$

where $\epsilon = 10^{-8}$ and β_1^k and β_2^k denote the k-th power of β_1 and β_2, respectively. α is the global stepsize and the recommended value is $\alpha = 0.001$. β_1 and β_2 are chosen to be 0.9 and 0.999, respectively [22].

2.4 Second Order Gradient Based Optimization

When it comes to highly non linear problems, first-order methods converge quite slowly and are often prone to getting stuck at local minima and flat regions. Incorporating second-order curvature information for training neural networks can improve convergence and avoid local minima, flat regions and saddle points. Second-order methods such as the Newton method, use both the gradient and second-order curvature information or the Hessian to determine the search direction. These methods typically exhibit quadratic rates of convergence.

2.4.1 Newton Methods

The second-order Taylor series of the objective function (2.1) around some point $\mathbf{w}_k + \mathbf{d}$ is given as

$$E(\mathbf{w}_k + \mathbf{d}) \approx m_k(\mathbf{d}) \approx E(\mathbf{w}_k) + \nabla E(\mathbf{w}_k)^{\mathrm{T}}\mathbf{d} + \frac{1}{2}\,\mathbf{d}^{\mathrm{T}}\nabla^2 E(\mathbf{w}_k)\mathbf{d}. \tag{2.22}$$

The Taylor approximation is minimized when $\nabla m_k(\mathbf{d}) = 0$, and thus we have the iteration scheme,

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k) \tag{2.23}$$

where $\mathbf{H}_k$ is the inverse of the Hessian matrix $\mathbf{B}_k = \nabla^2 E(\mathbf{w}_k) = \mathbf{H}_k^{-1}$ and α_k is the step size which is set to 1 or usually determined using line search or trust region method. This method of optimization, where we take into account the objective function's second-order behavior in addition to its first-order behavior, is known as Newton's method. At iteration k, the Newton method approximates the function at the point $\mathbf{w}_k$ with a paraboloid, and then proceeds to minimize that approximation by stepping to the minimum of that paraboloid. Figure 2.7 shows that the Newton method converged to the minimum in just one step. The main bottleneck in second-order methods is the serious computational challenges involved in the computation of the Hessian, $\nabla^2 E(\mathbf{w}_k)$ and its inverse, for large-scale problems, in which it is not practical because n is large.

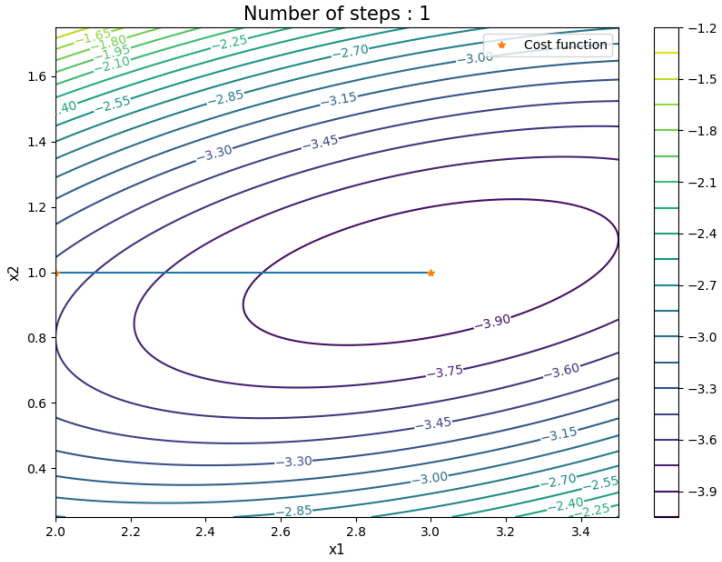


Figure 2.7: Evolution of the cost function during Newton method with level circles

2.4.2 Quasi-Newton Methods

Quasi-Newton methods form an alternative class of first-order methods for solving the large-scale nonconvex optimization problem in deep learning. These methods, like in SGD, require only computing the first-order gradient of the objective function. By measuring and storing the difference between consecutive gradients, quasi-Newton methods construct quasi-Newton matrices $\mathbf{B}_k$ which are low-rank updates to the previous Hessian approximations for estimating $\nabla^2 E(\mathbf{w}_k)$ at each iteration. They build a quadratic model of the objective function by using these quasi-Newton matrices and use that model to find a sequence of search directions that can result in superlinear convergence. Since these methods do not require the second-order derivatives, they are more efficient than Newton's method for large-scale optimization problems [1].

3

Quasi-Newton Methods for Training Neural Networks

Algorithms that seek to achieve some of the advantages of the Newton method without the computational complexity of the Hessian and its inverse are called quasi-Newton methods. This chapter discusses some of the quasi-Newton methods such as the SR1, BFGS and accelerated quasi-Newton (NAQ and MoQ) methods for training neural networks. The focus of this thesis is on the accelerated quasi-Newton method and hence this chapter serves to provide the necessary background for the chapters that follow. We discuss the effectiveness of these methods in comparison to popular first-order methods with examples on simple neural network structures for modeling function approximation problems and circuit modeling problems. The simulation examples discussed in this chapter are based on the results published in [27], [28], and [29].

3.1 Quasi-Newton Methods

We begin with the derivation of quasi-Newton methods with the quadratic model of the objective function at an iterate $\mathbf{w}_k$ given as

$$E(\mathbf{w}_k + \mathbf{d}) \approx m_k(\mathbf{d}) \approx E(\mathbf{w}_k) + \nabla E(\mathbf{w}_k)^{\mathrm{T}}\mathbf{d} + \frac{1}{2}\,\mathbf{d}^{\mathrm{T}}\nabla^2 E(\mathbf{w}_k)\mathbf{d}. \tag{3.1}$$

In order to find the minimizer $\mathbf{d}_k$, we equate $\nabla \mathbf{m}_k(\mathbf{d}) = 0$ and thus have

$$\mathbf{d}_k = -\nabla^2 E(\mathbf{w}_k)^{-1}\nabla E(\mathbf{w}_k) = -\mathbf{B}_k^{-1}\nabla E(\mathbf{w}_k)\,. \tag{3.2}$$

The new iterate $\mathbf{w}_{k+1}$ is given as,

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{B}_k^{-1}\nabla E(\mathbf{w}_k) = \mathbf{w}_k + \alpha_k \mathbf{d}_k, \tag{3.3}$$

and the quadratic model at the new iterate is given as

$$E(\mathbf{w}_{k+1} + \mathbf{d}) \approx m_{k+1}(\mathbf{d}) \approx E(\mathbf{w}_{k+1}) + \nabla E(\mathbf{w}_{k+1})^{\mathrm{T}}\mathbf{d} + \frac{1}{2}\,\mathbf{d}^{\mathrm{T}}\mathbf{B}_{k+1}\mathbf{d}\,, \tag{3.4}$$

where α_k is the step length and $\mathbf{B}_k^{-1} = \mathbf{H}_k$ and its consecutive updates $\mathbf{B}_{k+1}^{-1} = \mathbf{H}_{k+1}$ are symmetric positive definite matrices satisfying the quasi-Newton (or secant) condition.

Quasi-Newton methods are a generalization of the secant method, where the secant method can be thought of as a finite-difference approximation of Newton's method. Thus it is necessary for quasi-Newton methods to satisfy the secant condition. Thus, we require that the gradient of m_{k+1} should

match the gradient of the objective function at the last two iterates $\mathbf{w}_k$ and $\mathbf{w}_{k+1}$. In other words, we impose the following two requirements on $\mathbf{B}_{k+1}$,

$$\nabla m_{k+1}|_{\mathbf{d}=0} = \nabla E(\mathbf{w}_{k+1} + \mathbf{d})|_{\mathbf{d}=0} = \nabla E(\mathbf{w}_{k+1}), \tag{3.5}$$

$$\begin{aligned}\nabla m_{k+1}|_{\mathbf{d}=-\alpha_k \mathbf{d}_k} &= \nabla E(\mathbf{w}_{k+1} + \mathbf{d})|_{\mathbf{d}=-\alpha_k \mathbf{d}_k} \\ &= \nabla E(\mathbf{w}_{k+1} - \alpha_k \mathbf{d}_k) = \nabla E(\mathbf{w}_k).\end{aligned}$$

From (3.4),

$$\nabla m_{k+1}(\mathbf{d}) = \nabla E(\mathbf{w}_{k+1}) + \mathbf{B}_{k+1}\mathbf{d}. \tag{3.6}$$

Substituting $\mathbf{d} = 0$ in (3.6), the condition in (3.5) is satisfied. From (3.1) and substituting $\mathbf{d} = -\alpha_k \mathbf{d}_k$ in (3.6), we have

$$\nabla E(\mathbf{w}_k) = \nabla E(\mathbf{w}_{k+1}) - \alpha_k \mathbf{B}_{k+1}\mathbf{d}_k. \tag{3.7}$$

Substituting for $\alpha_k \mathbf{d}_k$ from (3.12) in (3.7), we get

$$\nabla E(\mathbf{w}_k) = \nabla E(\mathbf{w}_{k+1}) - \mathbf{B}_{k+1}(\mathbf{w}_{k+1} - \mathbf{w}_k). \tag{3.8}$$

On rearranging the terms, we have the secant or quasi-Newton condition.

$$\mathbf{y}_k = \mathbf{B}_{k+1}\mathbf{s}_k, \text{ or } \mathbf{s}_k = \mathbf{H}_{k+1}\mathbf{y}_k, \tag{3.9}$$

Thus from the secant condition we have

$$\mathbf{s}_k = \mathbf{w}_{k+1} - \mathbf{w}_k = \alpha_k \mathbf{d}_k \tag{3.10}$$

$$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k) \tag{3.11}$$

which are call the curvature information pair. Recall that the updates of quasi-Newton is given as

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k) \tag{3.12}$$

and the inverse Hessian $\mathbf{H}_k$ is iteratively updated using a low rank approximation satisfying the secant condition (3.9). Various quasi-Newton methods have been developed over the years, and they differ in how the approximate Hessian is updated at each iteration.

3.1.1 The SR1 Method

The Symmetric Rank-1 (SR1) method is a simple quasi-Newton method that uses a rank-one update for updating the Hessian approximation of the function being optimized [1]. The rank-one update of $\mathbf{B}_k$ is given by

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \sigma \mathbf{u}\mathbf{u}^{\mathrm{T}}, \tag{3.13}$$

where σ and $\mathbf{u}$ are chosen such that they satisfy the secant condtion in (3.9). Substituting (3.13) in (3.9), we get

$$\mathbf{y}_k = \mathbf{B}_k \mathbf{s}_k + (\sigma \mathbf{u}^{\mathrm{T}} \mathbf{s}_k)\mathbf{u}. \tag{3.14}$$

Since $(\sigma \mathbf{u}^{\mathrm{T}} \mathbf{s}_k)$ is a scalar, we can deduce $\mathbf{u}$ as a scalar multiple of $\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k$ and thus have

$$(\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k) = \sigma \delta^2 [\mathbf{s}_k^{\mathrm{T}} (\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)](\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k), \tag{3.15}$$

where

$$\sigma = \text{sign}[\mathbf{s}_k^{\mathrm{T}}(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)] \quad \text{and} \quad \delta = \pm|[\mathbf{s}_k^{\mathrm{T}}(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)]|^{1/2} . \tag{3.16}$$

Thus the symmetric rank-1 (SR1) update of the Hessian is given as

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)^{\mathrm{T}}}{(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)^{\mathrm{T}}\mathbf{s}_k} . \tag{3.17}$$

By applying the Sherman-Morrison-Woodbury formula (see Appendix A.2), we can find $\mathbf{B}_{k+1}^{-1} = \mathbf{H}_{k+1}^{\mathrm{SR1}}$ as

$$\mathbf{H}_{k+1}^{\mathrm{SR1}} = \mathbf{H}_k^{\mathrm{SR1}} + \frac{(\mathbf{s}_k - \mathbf{H}_k^{\mathrm{SR1}}\mathbf{y}_k)(\mathbf{s}_k - \mathbf{H}_k^{\mathrm{SR1}}\mathbf{y}_k)^{\mathrm{T}}}{(\mathbf{s}_k - \mathbf{H}_k^{\mathrm{SR1}}\mathbf{y}_k)^{\mathrm{T}}\mathbf{y}_k}, \tag{3.18}$$

where,

$$\mathbf{s}_k = \mathbf{w}_{k+1} - \mathbf{w}_k \text{ and } \mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k) \tag{3.19}$$

The main drawback of SR1 updating is that the denominator in (3.18) may vanish and that it does not guarantee positive definiteness. However, the matrices generated by the SR1 formula tend to be good approximations to the true Hessian matrix [1]. Further, SR1 with trust region approach is known to guarantee positive definiteness and have bounded Hessian approximations that guarantees convergence. Trust-region methods attempt to find the search direction, in a region within which they trust the accuracy of the quadratic model of the objective function. The SR1 algorithm using trust region approach is as shown in Algorithm 3.1.

Algorithm 3.1 SR1 Method

while $\|\nabla E(\mathbf{w}_k)\| > \epsilon$ and $k < k_{\max}$ **do**
  Compute $\nabla E(\mathbf{w}_k)$
  Find $\mathbf{s}_k$ by solving the subproblem
    $\min_{\mathbf{s}} \ \nabla E(\mathbf{w}_k)^{\mathrm{T}}\mathbf{s} + \frac{1}{2}\mathbf{s}^{\mathrm{T}}\mathbf{B}_k\mathbf{s}, \quad \text{subject to } \|s\| \leq \Delta_k$
  Compute $\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k)$
  Compute $\rho_k = \frac{E(\mathbf{w}_k) - E(\mathbf{w}_k + \mathbf{s}_k)}{m_k(0) - m_k(\mathbf{s}_k)}$
  if $\rho_k \geq \eta$ **then**
    Set $\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_{k+1}$
  else
    Set $\mathbf{w}_{k+1} = \mathbf{w}_k$
  end if
  $\Delta_{k+1} = \text{adjustTR}(\Delta_k, \rho_k)$
  if $|\mathbf{s}_k^{\mathrm{T}}(\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k)| \geq \epsilon \ \|\mathbf{s}_k\| \ \|\mathbf{y}_k - \mathbf{B}_k\mathbf{s}_k\|$ **then**
    Update $\mathbf{B}_k$ using (3.18)
  else
    $\mathbf{B}_{k+1} \leftarrow \mathbf{B}_k$
  end if
end while

3.1.2 The BFGS quasi-Newton Method

SR1 updates are easy to use, but they have the disadvantage that the Hessian matrix $\mathbf{B}_k$ is not guaranteed to be positive definite, and thus the search direction $\mathbf{g}_k$ is not guaranteed to be a descent direction. The Broyden-Fletcher-Goldfarb-Shanon (BFGS) algorithm [1] is one of the most popular quasi-Newton

methods that overcomes the weaknesses of SR1. It is by far the most successful quasi-Newton method for unconstrained optimization due to its computational efficiency and good asymptotic convergence. In the BFGS method, the Hessian $\mathbf{B}_k$ is updated using a symmetric rank-2 matrix as shown below.

$$\mathbf{B}_{k+1} = \mathbf{B}_k + a\mathbf{u}\mathbf{u}^{\mathrm{T}} + b\mathbf{v}\mathbf{v}^{\mathrm{T}} \tag{3.20}$$

where $\mathbf{u}$ and $\mathbf{v}$ are linearly independent non-zero vectors and a and b are constants, chosen to satisfy the secant condition.

$$\mathbf{B}_{k+1}\mathbf{s}_k = \mathbf{B}_k\mathbf{s}_k + a\mathbf{u}\mathbf{u}^{\mathrm{T}}\mathbf{s}_k + b\mathbf{v}\mathbf{v}^{\mathrm{T}}\mathbf{s}_k = \mathbf{y}_k. \tag{3.21}$$

A natural choice for $\mathbf{u}$ and $\mathbf{v}$ would be $\mathbf{u} = \mathbf{y}_k$ and $\mathbf{v} = \mathbf{B}_k\mathbf{s}_k$. We thus have,

$$\mathbf{B}_k\mathbf{s}_k + a\mathbf{y}_k\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k + b\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}\mathbf{B}_k^T\mathbf{s}_k = \mathbf{y}_k. \tag{3.22}$$

$$\mathbf{B}_k\mathbf{s}_k(1 + b\mathbf{s}_k^{\mathrm{T}}\mathbf{B}_k^T\mathbf{s}_k) = \mathbf{y}_k(1 - a\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k). \tag{3.23}$$

$$\Rightarrow \quad a = \frac{1}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}, \quad b = -\frac{1}{\mathbf{s}_k^{\mathrm{T}}\mathbf{B}_k^T\mathbf{s}_k} \tag{3.24}$$

Substituting for a, b, $\mathbf{u}$ and $\mathbf{v}$ in (3.20), we have

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k\mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k} - \frac{\mathbf{B}_k\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}\mathbf{B}_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}}\mathbf{B}_k^T\mathbf{s}_k} \tag{3.25}$$

The inverse of the Hessian matrix $\mathbf{B}_k^{-1} = \mathbf{H}_k^{\mathrm{BFGS}}$ can be obtained using the Sherman-Morrison-Woodbury formula (Appendix A.2) as

$$\mathbf{H}_{k+1}^{\mathrm{BFGS}} = \left(\mathbf{I} - \frac{\mathbf{y}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\right)\mathbf{H}_k^{\mathrm{BFGS}}\left(\mathbf{I} - \frac{\mathbf{y}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\right) + \frac{\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}, \tag{3.26}$$

where $\mathbf{I}$ denotes identity matrix. Thus the weight update using the BFGS formula is given as

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k\mathbf{H}_k^{\mathrm{BFGS}}\nabla E(\mathbf{w}_k), \tag{3.27}$$

where $\mathbf{H}_k^{\mathrm{BFGS}}$ is updated using (3.26) with

$$\mathbf{s}_k = \mathbf{w}_{k+1} - \mathbf{w}_k \ \text{ and } \ \mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k) \tag{3.28}$$

and α_k is the step size which is usually determined by a line search procedure such as Armijo or Wolfe search. The Armijo line search condition is as shown below.

$$E(\mathbf{w}_k + \alpha_k\mathbf{g}_k) \leq E(\mathbf{w}_k) + \eta\alpha_k\nabla E(\mathbf{w}_k)^{\mathrm{T}}\mathbf{g}_k, \tag{3.29}$$

The BFGS quasi-Newton algorithm is shown in Algorithm 3.2.

3.1.3 Limited Memory BFGS Method

The limited memory BFGS (LBFGS), is a variant of the BFGS quasi-Newton method, designed for solving large-scale optimization problems such as training deep neural network models. As the scale of the neural network model increases, the $O(d^2)$ cost of storing and updating the Hessian matrix $\mathbf{H}_k^{\mathrm{BFGS}}$

Algorithm 3.2 BFGS Method

Require: ε and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$ and $\mathbf{H}_k^{\text{BFGS}} = \mathbf{I}$.
1: $k \leftarrow 1$
2: Calculate $\nabla E(\mathbf{w}_k)$
3: **while** $\|E(\mathbf{w}_k)\| > \varepsilon$ and $k < k_{max}$ **do**
4: $\quad \mathbf{g}_k \leftarrow -\mathbf{H}_k^{\text{BFGS}} \nabla E(\mathbf{w}_k)$
5: $\quad$ Determine α_k by line search
6: $\quad \mathbf{v}_{k+1} \leftarrow \alpha_k \mathbf{g}_k$
7: $\quad \mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
8: $\quad$ Calculate $\nabla E(\mathbf{w}_{k+1})$
9: $\quad$ Update $\mathbf{H}_{k+1}^{\text{BFGS}}$ using (3.26)
10: $\quad k \leftarrow k + 1$
11: **end while**

is expensive. In the limited memory version, the Hessian matrix is defined by recursively applying m number of BFGS updates using only the last m number of curvature pairs $\{\mathbf{s}_k, \mathbf{y}_k\}$ as

$$\begin{aligned}
\mathbf{H}_k^{\text{BFGS}} = {} & (\mathbf{V}_{k-1}^{\mathrm{T}} \cdots \mathbf{V}_{k-m}^{\mathrm{T}})\, \mathbf{H}_k^0\, (\mathbf{V}_{k-m} \cdots \mathbf{V}_{k-m}) \\
& + \frac{1}{\mathbf{y}_{k-m}^{\mathrm{T}} \mathbf{s}_{k-m}} (\mathbf{V}_{k-1}^{\mathrm{T}} \cdots \mathbf{V}_{k-m+1}^{\mathrm{T}})\, \mathbf{s}_{k-m} \mathbf{s}_{k-m}^{\mathrm{T}}\, (\mathbf{V}_{k-m+1} \cdots \mathbf{V}_{k-m+1}) \\
& + \frac{1}{\mathbf{y}_{k-m+1}^{\mathrm{T}} \mathbf{s}_{k-m+1}} (\mathbf{V}_{k-1}^{\mathrm{T}} \cdots \mathbf{V}_{k-m+2}^{\mathrm{T}})\, \mathbf{s}_{k-m+1} \mathbf{s}_{k-m+1}^{\mathrm{T}}\, (\mathbf{V}_{k-m+2} \cdots \mathbf{V}_{k-m+2}) \\
& \vdots \\
& + \frac{1}{\mathbf{y}_{k-1}^{\mathrm{T}} \mathbf{s}_{k-1}} \mathbf{s}_{k-1} \mathbf{s}_{k-1}^{\mathrm{T}}
\end{aligned} \tag{3.30}$$

As a result, the computational cost is significantly reduced and the storage cost is down to $O(md)$ where d is the number of parameters and m is the memory size. The search direction $\mathbf{g}_k = -\mathbf{H}_k^{\text{BFGS}} \nabla E(\mathbf{w}_k)$ is effectively implemented using the two-loop recursion as shown in Appendix A.1.

3.2 Nesterov's Acceleration

Nesterov acceleration is an extension of momentum that involves calculating the decaying moving average of the gradients of projected positions in the search space rather than the actual positions themselves. This has the effect of harnessing the accelerating benefits of momentum whilst allowing the search to slow down when approaching the optima and reduce the likelihood of missing or overshooting it. The Nesterov's acceleration was introduced by Yuri Nesterov in [20] in order to speed up the convergence of gradient descent method. [19] popularized the Nesterov's acceleration in training neural networks. The Nesterov's Accelerated Gradient (NAG) achieves a global convergence rate of $O(1/k^2)$ versus the $O(1/k)$ of the gradient descent method. The NAG update can is given as

$$\mathbf{v}_{k+1} = \mu \mathbf{v}_k - \alpha_k \nabla E(\mathbf{w}_k + \mu \mathbf{v}_k) \tag{3.31}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_{k+1} \tag{3.32}$$

Intuitively, the Nesterov's acceleration can be thought of as a modification to the momentum method to overcome overshooting the minima. It explores points near the current iterate got from the gradient descent-type algorithm, and chooses larger and larger exploratory steps (as governed by the increasing size of the momentum term). But despite this exploration, the Nesterov's acceleration does not diverge i.e. even if a particular iteration overshoots and moves far from the optimum, it will get back close to the optimum after a few more iterations [30].

In the following section, we shall discuss the second-order methods that apply the concept of Nesterov's acceleration in quasi-Newton updates.

3.3 Accelerated Quasi-Newton Methods

3.3.1 The NAQ Method

Several modifications have been proposed to the quasi-Newton method to obtain stronger convergence. The Nesterov's Accelerated Quasi-Newton (NAQ) [31] method achieves faster convergence compared to the standard BFGS quasi-Newton methods by quadratic approximation of the objective function at $\mathbf{w}_k + \mu\mathbf{v}_k$ and by incorporating the Nesterov's accelerated gradient $\nabla E(\mathbf{w}_k + \mu\mathbf{v}_k)$ in its Hessian update. The derivation of NAQ is briefly discussed as follows.

Let $\mathbf{d}$ be the vector $\mathbf{d} = \mathbf{w} - \left(\mathbf{w}_k + \mu\mathbf{v}_k\right)$. The quadratic approximation of the objective function at $\mathbf{w}_k + \mu\mathbf{v}_k$ is defined as,

$$E(\mathbf{w}) \simeq E(\mathbf{w}_k + \mu\mathbf{v}_k) + \nabla E(\mathbf{w}_k + \mu\mathbf{v}_k)^{\mathrm{T}}\Delta\mathbf{w} + \frac{1}{2}\mathbf{d}^{\mathrm{T}}\nabla^2 E(\mathbf{w}_k + \mu\mathbf{v}_k)\mathbf{d}. \tag{3.33}$$

The minimizer of this quadratic function is explicitly given by

$$\mathbf{d} = -\nabla^2 E\left(\mathbf{w}_k + \mu\mathbf{v}_k\right)^{-1} \nabla E\left(\mathbf{w}_k + \mu\mathbf{v}_k\right). \tag{3.34}$$

Therefore the new iterate is defined as

$$\mathbf{w}_{k+1} = \left(\mathbf{w}_k + \mu\mathbf{v}_k\right) - \nabla^2 E\left(\mathbf{w}_k + \mu\mathbf{v}_k\right)^{-1} \nabla E\left(\mathbf{w}_k + \mu\mathbf{v}_k\right). \tag{3.35}$$

This iteration can be considered as Newton method with the momentum term $\mu\mathbf{v}_k$. The inverse of Hessian $\nabla^2 E(\mathbf{w}_k + \mu\mathbf{v}_k)$ is approximated by the matrix $\mathbf{H}_{k+1}^{\mathrm{NAQ}}$ using the following update equation,

$$\mathbf{H}_{k+1}^{\mathrm{NAQ}} = \left(\mathbf{I} - \frac{\mathbf{s}_k\mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\right)\mathbf{H}_k^{\mathrm{NAQ}}\left(\mathbf{I} - \frac{\mathbf{y}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}\right) + \frac{\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k}, \tag{3.36}$$

where

$$\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu\mathbf{v}_k) \text{ and } \mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k + \mu\mathbf{v}_k). \tag{3.37}$$

(3.36) is derived from the secant condition (3.9) and the rank-2 updating formula [31]. It is proved that the Hessian matrix $\mathbf{H}_{k+1}^{\mathrm{NAQ}}$ updated by (3.36) is a positive definite symmetric matrix given $\mathbf{H}_k^{\mathrm{NAQ}}$ is positive definite and $\mathbf{H}_0^{\mathrm{NAQ}}$ is initialized to identity matrix [31]. Therefore, the update vector of NAQ can be written as:

$$\mathbf{v}_{k+1} = \mu\mathbf{v}_k + \alpha_k\mathbf{g}_k, \tag{3.38}$$

where $\mathbf{g}_k = -\mathbf{H}_k^{\text{NAQ}} \nabla E(\mathbf{w}_k + \mu \mathbf{v}_k)$ is the search direction. The NAQ algorithm is given in Algorithm 3.3. Note that the gradient is computed twice in one iteration. This increases the computational cost compared to the BFGS quasi-Newton method. However, due to acceleration by the momentum and Nesterov's gradient term, NAQ is faster in convergence compared to BFGS.

3.3.2 Limited Memory NAQ

Similar to the LBFGS method, LNAQ [32] is the limited memory variant of NAQ that uses the last m curvature pairs $\{\mathbf{s}_k, \mathbf{y}_k\}$. In the limited-memory form note that the curvature pairs that are used incorporate the momemtum and Nesterov's accelerated gradient term, thus accelerating LBFGS. Implementation of LNAQ algorithm can be realized by omitting steps 4 and 9 of Algorithm 3.3 and determining the search direction $\mathbf{g}_k$ using the two-loop recursion shown in Appendix A.1. The last m vectors of $\mathbf{s}_k$ and $\mathbf{y}_k$ are stored and used in the direction update.

Algorithm 3.3 NAQ Method

Require: $0 < \mu < 1$, ε and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$, $\mathbf{H}_k = \mathbf{I}$ and $\mathbf{v}_k = 0$.
1: $k \leftarrow 1$
2: **while** $\|E(\mathbf{w}_k)\| > \varepsilon$ and $k < k_{max}$ **do**
3: Calculate $\nabla E(\mathbf{w}_k + \mu \mathbf{v}_k)$
4: $\mathbf{g}_k \leftarrow -\mathbf{H}_k^{\text{NAQ}} \nabla E(\mathbf{w}_k + \mu \mathbf{v}_k)$
5: Determine α_k by line search
6: $\mathbf{v}_{k+1} \leftarrow \mu \mathbf{v}_k + \alpha_k \mathbf{g}_k$
7: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
8: Calculate $\nabla E(\mathbf{w}_{k+1})$
9: Update $\mathbf{H}_k^{\text{NAQ}}$ using (3.36)
10: $k \leftarrow k + 1$
11: **end while**

3.3.3 The MoQ Method

The Momentum quasi-Newton (MoQ) method [33] is realized by approximating the Nesterov's accelerated gradient vector as a linear combination of the gradients at the current and previous iterates, thus resulting in computing only one gradient per iteration. MoQ approximates the error function $E(\mathbf{w})$ by assuming that the function is approximately quadratic in the neighborhood of $\mathbf{w}_k + \mu \mathbf{v}_k$. Thus the Nesterov's gradient term can be approximated as,

$$\nabla E(\mathbf{w}_k + \mu \mathbf{v}_k) \simeq \nabla E(\mathbf{w}_k) + \mu \nabla E(\mathbf{v}_k). \tag{3.39}$$

Furthermore, since $\mathbf{v}_k = \mathbf{w}_k - \mathbf{w}_{k-1}$, (3.39) can be rewritten as

$$\begin{aligned} \nabla E(\mathbf{w}_k + \mu \mathbf{v}_k) &\simeq \nabla E(\mathbf{w}_k) + \mu \nabla E(\mathbf{w}_k - \mathbf{w}_{k-1}) \\ &\simeq (1 + \mu) \nabla E(\mathbf{w}_k) - \mu \nabla E(\mathbf{w}_{k-1}). \end{aligned} \tag{3.40}$$

From (3.39) and (3.40), it is confirmed that Nesterov's accelerated gradient can be approximated as a weighted linear combination of $\nabla E(\mathbf{w}_k)$ and $\nabla E(\mathbf{w}_{k-1})$ with a momentum coefficient μ. Therefore,

MoQ can be regarded as a method accelerating the BFGS method using the momentum term. The update vector of MoQ is given as

$$\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k + \alpha_k \mathbf{g}_k, \tag{3.41}$$

where $\mathbf{g}_k$ is the search direction given by

$$\mathbf{g}_k = \mathbf{H}_k^{\text{MoQ}} \nabla \mathbf{E}(\mathbf{w}_k + \mu_k \mathbf{v}_k). \tag{3.42}$$

From (3.40), the search direction can be approximated as

$$\mathbf{g}_k = \mathbf{H}_k^{\text{MoQ}} [(1 + \mu_k)\nabla \mathbf{E}(\mathbf{w}_k) - \mu_k \nabla \mathbf{E}(\mathbf{w}_{k-1})]. \tag{3.43}$$

where $\mathbf{H}_k^{\text{MoQ}}$ is updated by,

$$\mathbf{H}_{k+1}^{\text{MoQ}} = \left(\mathbf{I} - \frac{\mathbf{s}_k \hat{\mathbf{y}}_k^{\text{T}}}{\hat{\mathbf{y}}_k^{\text{T}} \mathbf{s}_k}\right) \mathbf{H}_k^{\text{MoQ}} \left(\mathbf{I} - \frac{\hat{\mathbf{y}}_k \mathbf{s}_k^{\text{T}}}{\hat{\mathbf{y}}_k^{\text{T}} \mathbf{s}_k}\right) + \frac{\mathbf{s}_k \mathbf{s}_k^{\text{T}}}{\hat{\mathbf{y}}_k^{\text{T}} \mathbf{s}_k}, \tag{3.44}$$

with the curvature information pair $\{\mathbf{s}_k, \hat{\mathbf{y}}_k\}$ as

$$\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu \mathbf{v}_k) = \mathbf{w}_{k+1} - (1 + \mu)\mathbf{w}_k + \mu \mathbf{w}_{k-1}, \tag{3.45}$$

$$\hat{\mathbf{y}}_k = \nabla E(\mathbf{w}_{k+1}) - (1 + \mu)\nabla E(\mathbf{w}_k) + \mu \nabla E(\mathbf{w}_{k-1}). \tag{3.46}$$

The algorithm of MoQ is as shown in Algorithm 3.4.

Algorithm 3.4 MoQ Method

Require: $0 < \mu < 1$, ε and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$, $\mathbf{H}_k = \mathbf{I}$ and $\mathbf{v}_k = 0$.
1: $k \leftarrow 1$
2: Calculate $\nabla E(\mathbf{w}_k)$
3: **while** $\|E(\mathbf{w}_k)\| > \varepsilon$ and $k < k_{max}$ **do**
4: $\quad \mathbf{g}_k \leftarrow -\mathbf{H}_k^{\text{MoQ}} [(1 + \mu_k)\nabla \mathbf{E}(\mathbf{w}_k) - \mu_k \nabla \mathbf{E}(\mathbf{w}_{k-1})]$
5: $\quad$ Determine α_k by line search
6: $\quad \mathbf{v}_{k+1} \leftarrow \mu \mathbf{v}_k + \alpha_k \mathbf{g}_k$
7: $\quad \mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
8: $\quad$ Store $\nabla \mathbf{E}(\mathbf{w}_k)$ and calculate $\nabla \mathbf{E}(\mathbf{w}_{k+1})$
9: $\quad$ Update $\mathbf{H}_k^{\text{MoQ}}$ using (3.44)
10: $\quad k \leftarrow k + 1$
11: **end while**

3.3.4 Limited Memory MoQ

Similar to the LBFGS and LNAQ methods, LMoQ [28] is the limited memory variant of the MoQ that uses the last m curvature pairs $\{\mathbf{s}_k, \hat{\mathbf{y}}_k\}$. Implementation of LNAQ algorithm can be realized by omitting steps 4 and 9 of Algorithm 3.4. and determining the search direction $\mathbf{g}_k$ using the two-loop recursion shown in Appendix A.1.

3.4 Simulation Examples

Now that we have discussed some of the commonly used first and second-order quasi-Newton methods, we shall discuss the implementation details and their performance evaluation when used for training neural networks. We conduct a performance evaluation of the methods discussed above on a set of non-convex function approximation problems and electronic circuit and design problems trained in full batch. The simulations are performed on the Tensorflow framework. The first-order AdaGrad, RMSProp, Adam methods are called using the Tensorflow's built-in optimizer implementations. The hyper-parameters of AdaGrad, RMSProp and Adam are set to their default values. The BFGS method is implemented in the Scipy library and used in Tensorflow v1.x through the ScipyOptimizerInterface class. Similarly we implemented NAQ and MoQ methods in the Scipy library and are called in Tensorflow v1.x using the ScipyOptimizerInterface class. In order to guarantee numerical stability and better convergence, an additional $\hat{\xi}_k \mathbf{s}_k$ term was added to $\mathbf{y}_k$ [27]. Thus, the vector $\mathbf{y}_k$ in the modified method is given as

$$\mathbf{y}_k = \mathbf{y}_k + \hat{\xi}_k \mathbf{s}_k. \tag{3.47}$$

where ξ_k is defined as

$$\hat{\xi}_k = \omega \parallel \nabla E(\mathbf{w}_k) \parallel + max\{-\epsilon_k^{\mathrm{T}} \mathbf{s}_k / \parallel \mathbf{s}_k \parallel^2, 0\}, \tag{3.48}$$

$$\begin{cases} \omega = 2 & if \parallel \nabla E(\mathbf{w}_k) \parallel^2 > 10^{-2}, \\ \omega = 100 & if \parallel \nabla E(\mathbf{w}_k) \parallel^2 < 10^{-2}. \end{cases} \tag{3.49}$$

In case of the NAQ and MoQ methods, $\mathbf{w}_k$ is replaced by $\mathbf{w}_k + \mu \mathbf{v}_k$ and the corresponding MoQ approximation.

We begin with evaluating the full memory BFGS and NAQ method in comparison to first-order methods. It was observed that in examples considered for full memory implementaion, the linesearch method failed to optimize. Hence, we used an explicit formula for determining the stepsize α_k [27].

$$\alpha_k = -\frac{\delta \nabla E(\mathbf{w}_k)^{\mathrm{T}} \mathbf{g}_k}{\|\mathbf{g}_k\|_{Q_k}^2}, \tag{3.50}$$

where

$$\|\mathbf{g}_k\|_{Q_k}^2 = \sqrt{\mathbf{g}_k^{\mathrm{T}} Q_k \mathbf{g}_k}. \tag{3.51}$$

Q_k is determined by $Q_k = L\mathbf{I}$ where L is the Lipschitz constant of the gradient. L is chosen to be $L = 100\|\mathbf{y}_k\|/\|\mathbf{s}_k\|$. In case of the NAQ method, $\mathbf{w}_k$ in (3.50) is replaced by $\mathbf{w}_k + \mu \mathbf{v}_k$. The BFGS and NAQ methods implementing this stepsize is denoted as mBFGS and mNAQ respectively.

3.4.1 Sinusoidal function approximation problem

The function approximation problem under consideration is given as

$$f(a, x, b) = 1 + (x + 2x^2)\sin(-ax^2 + b), \qquad |x| \leq 4. \tag{3.52}$$

Consider the case where $a = -1$ and $b = 0$. Thus, the function reduces to a single input function in x given by $f(x) = 1 + (x + 2x^2)\sin(-x^2)$. The training samples are generated with an interval of 0.02 while the test samples are generated by random sampling in the range $x \in [-4, 4)$. Each element

Table 3.1: Summary of simulation results of function approximation problem.

Algorithm	μ	$E(\mathbf{w})(\times 10^{-3})$ Ave/Best/Worst	Time (s)	Iteration count	$E_{test}(\mathbf{w})(\times 10^{-3})$ Ave/Best/Worst
AdaGrad	-	59.8 / 58.6 / 60.2	40	100,000	59.03 / 57.69 / 59.48
RMSprop	-	3.34 / 0.564 / 7.89	41	100,000	3.35 / 0.409 / 8.16
Adam	-	4.15 / 0.324 / 14.3	42	100,000	4.14 / 0.359 / 14.53
BFGS	-	15.14 / 0.650 / 31.80	4.9	3,204	15.14 / 0.650 / 30.66
mBFGS	-	5.24 / 0.194 / 17.8	58	31,370	5.26 / 0.233 / 17.80
mNAQ	0.8	1.94 / 0.307 / 6.33	23	9,006	1.94 / 0.307 / 6.33
	0.85	0.974 / 0.307 / 5.00	19	7,549	0.980 / 0.315 / 5.00
	0.9	1.53 / 0.194 / 13.8	15	5,931	1.53 / 0.194 / 13.80
	0.95	1.30 / 0.195 / 6.31	11	4,461	1.30 / 0.233 / 6.31

of the input and desired outputs of the training and test data are normalized in the range$[-1, 1]$. The training and test set consists of 400 and 10000 samples respectively. The number of hidden neurons used is 7. Thus, the neural network structure is given as 1-7-1 with sigmoid activation function. The maximum number of iterations k_{max} is chosen to be $k_{max} = 100,000$ and the terminate condition is set to $\epsilon = 1.0 \times 10^{-6}$. The parameter σ is chosen to be $\sigma = 10^{-3}$. We conduct 15 independent and the summary of the results is presented in Table 3.1. NAQ failed to determine a suitable step size

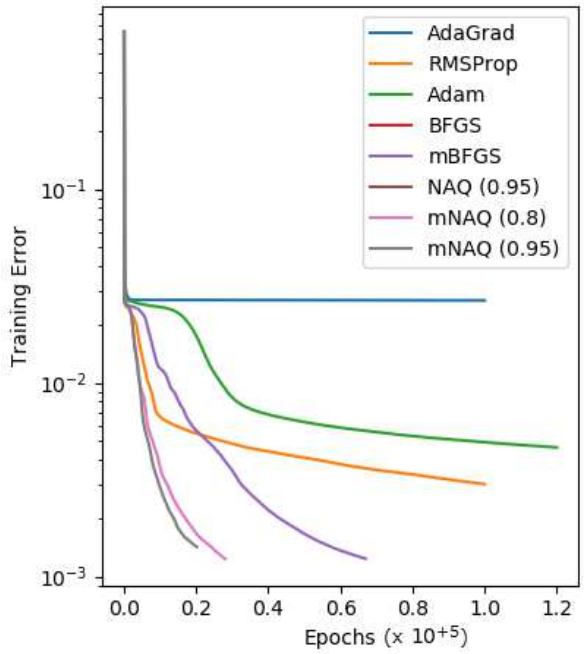


Figure 3.1: The average training errors *vs* iteration count

and hence terminated much earlier without converging. Thus, the corresponding results are omitted from the table. The results indicate that the second-order methods mBFGS and mNAQ converge faster with smaller errors compared to the first-order algorithms. On comparing the second-order methods, mBFGS results are comparable with mNAQ. However, it is 12 times slower and takes almost 9.9 times more number of epochs to converge (Figure 3.1). The mNAQ algorithm results in 5-7 times smaller error rates compared to BFGS. On comparing the results of mNAQ with different values of the momentum term, $\mu = 0.95$ is the fastest with least number of average epochs while 0.85 has the least

average training error. Figure 3.2 illustrates the output of the function under consideration versus the output of the trained neural network using mBFGS and mNAQ with a momentum of $\mu = 0.85$. The output of the neural network trained with mNAQ is in close approximation with the original function output, confirming its accuracy.

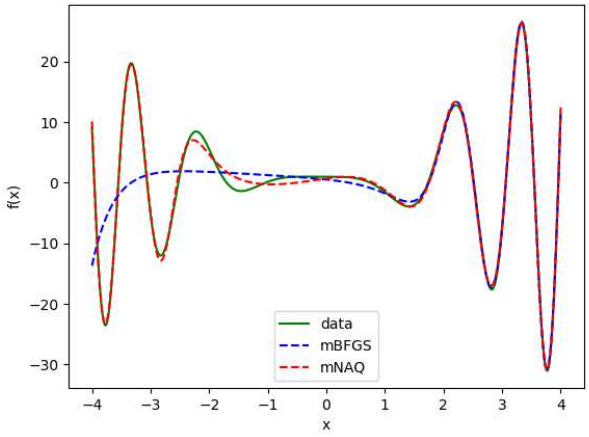


Figure 3.2: Comparison of network models of mBFGS and mNAQ *vs* original test data

3.4.2 Levy function approximation problem

Next we evaluate the limited memory quasi-Newton methods on the Levy function approximation problem which is defined in (3.53).

$$f(x_1 \dots x_n) = \frac{\pi}{n}\Big\{ \sum_{i=1}^{n-1}[(x_i - 1)^2(1 + 10\sin^2(\pi x_{i+1}))] + 10\sin^2(\pi x_1) + (x_n - 1)^2 \Big\}, x_i \in [-4, 4], \forall i. \tag{3.53}$$

A feedforward neural network with one hidden layer of 50 hidden neurons with sigmoid activation function is used. Therefore the neural network structure is $5 - 50 - 1$. The number of parameters is $d = 351$. Mean squared errors function was used and the maximum number of iteration is set to $k_{\max} = 10000$, terminate condition $\epsilon = 10^{-6}$ and limited memory size $m = 16$. The step size of the limited memory quasi-Newton methods are determined by applying backtracking line search that satisfies the Armijo's condition as given in equation (3.4.2).

$$\begin{aligned} E(\mathbf{w}_k + \alpha_k \hat{\mathbf{g}}_k) &\leq E(\mathbf{w}_k) + \eta\alpha_k \nabla E(\mathbf{w}_k)^{\mathrm{T}} \hat{\mathbf{g}}_k && \text{for LBFGS,} \\ E(\mathbf{w}_k + \mu\mathbf{v}_k + \alpha_k \hat{\mathbf{g}}_k) &\leq E(\mathbf{w}_k + \mu\mathbf{v}_k) + \eta\alpha_k \nabla E(\mathbf{w}_k + \mu\mathbf{v}_k)^{\mathrm{T}} \hat{\mathbf{g}}_k && \text{for LNAQ / LMoQ,} \end{aligned} \tag{3.54}$$

where $0 < \eta < 1$ and default value is $\eta = 0.001$.

Also, we evaluate the performance with an adaptive momentum scheme [28] where the μ_k value is updated using

$$\mu_k = \theta_k(1 - \theta_k)/({\theta_k}^2 + \theta_{k+1}), \tag{3.55}$$

Table 3.2: Summary of results on the Levy function approximation problem, averaged over 50 trials.

Method	$E(\mathbf{w})$	epochs	fev	gev	time(s)
L-BFGS	0.000091	10000	28398	10001	81.95
L-NAQ	0.000025	9927	20848	19854	95.21
L-MoQ	**0.000022**	**9961**	**20918**	**9962**	**73.73**

where θ_{k+1} is obtained by solving (3.56) with $\theta_0 = 1$ and $\gamma = 10^{-5}$.

$$\theta_{k+1}{}^2 = (1 + \theta_{k+1})\theta_k{}^2 + \gamma\theta_{k+1}. \tag{3.56}$$

Table 3.2 and Figure 3.3 show the average results over 50 independent trials. The number of function and gradient evaluations are denoted as fev and gev, respectively. From the error vs. iterations plot, we can confirm that the L-MoQ method is a good approximation to L-NAQ since they are in close approximation to each other. However from the error vs. time plot it is clear that L-MoQ converges faster compared to L-NAQ, which is due to the computation of only one gradient per iteration, similar to that of the L-BFGS method. Thus L-MoQ has fewer gradient evaluations gev too, and maintains the same computational cost of L-BFGS i.e., $nd + 4md + 2d + \zeta nd$ and storage cost of $(2m + 1)d$. But due to the acceleration effect of the momentum term, L-MoQ converges faster than L-BFGS.

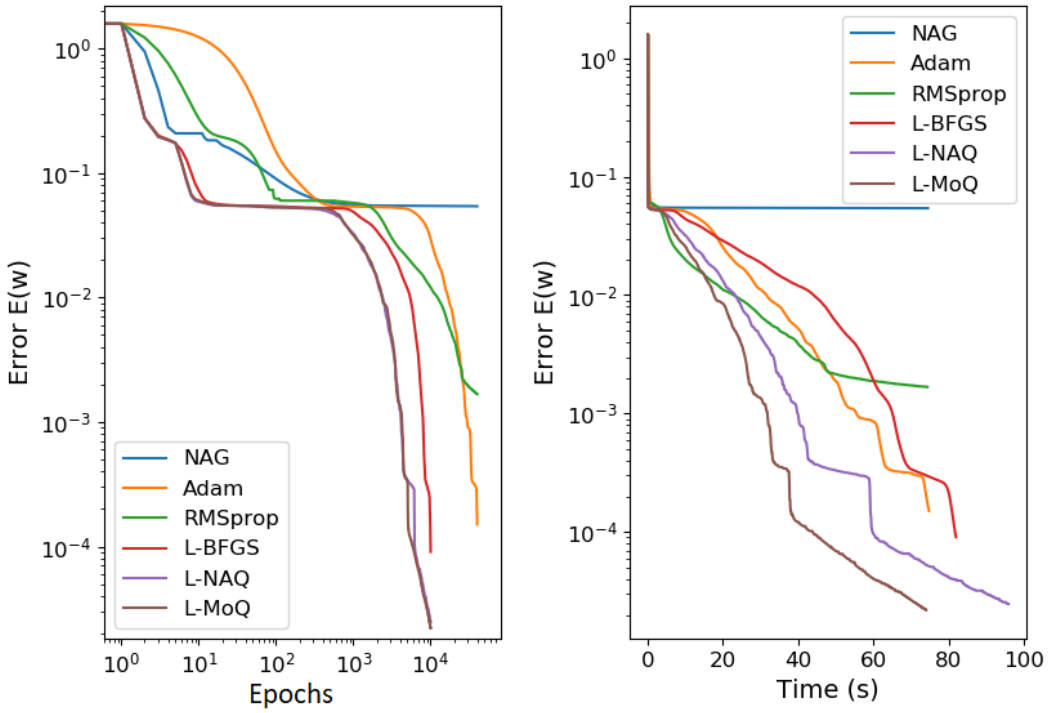


Figure 3.3: Results on the Levy function approximation problem, averaged over 50 trials.

3.4.3 Microstrip low pass filter modeling problem

The performance of the common first-order and second-order methods are evaluated on a large microwave circuit problem to model a microstrip low pass filter (LPF) [27]. The dielectric constant and

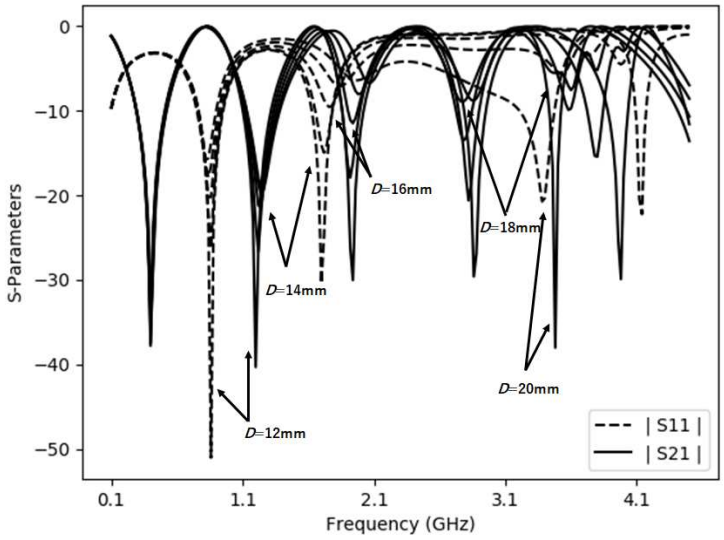


Figure 3.4: Training data set of microstrip lowpass filter (LPF).

height of the substrate of the LPF are 9.3 and 1mm, respectively. The inputs to the neural network are the length D and frequency f. The outputs are the magnitudes of the S-parameters $|S_{11}|$ and $|S_{21}|$.The frequency range is 0.1 to 4.5 GHz. For the training and test data, length D ranges between 12-20 mm and 13-19 mm respectively at intervals of 2mm. Each interval contains 221 samples. The training set comprises of 1105 samples and test set comprises 884 samples. The training and test data were generated using Sonnet [34]. The number of hidden neurons used is 45. Figure 3.4 shows the training data of the microstrip low pass filter.

Table 3.3: Summary of simulation results of microstrip low-pass filter (LPF).

Algorithm	μ	$E(\mathbf{w})(\times 10^{-3})$ Ave/Best/Worst	Time (s)	Iteration count	$E_{test}(\mathbf{w})(\times 10^{-3})$ Ave/Best/Worst
AdaGrad	-	26.6 / 26.4 / 26.7	112	100,000	22.4 / 22.3 / 22.5
RMSprop	-	2.99 / 2.44 / 4.07	113	100,000	7.00 / 1.88 / 36.0
Adam	-	4.63 / 3.67 / 5.60	137	100,000	37.0 / 3.41 / 212.5
mBFGS	-	1.04 / 0.834 / 1.46	493	81,457	1.01 / 0.529 / 3.52
mNAQ	0.8	0.93 / 0.827 / 1.37	303	38,470	0.744 / 0.534 / 1.07
	0.85	1.02 / 0.756 / 1.62	314	39,678	7.32 / 5.75 / 87.8
	0.9	1.00 / 0.716 / 1.46	242	30,619	0.842 / 0.558 / 1.87
	0.95	1.24 / 0.834 / 1.85	209	26,547	2.08 / 0.600 / 13.7

Table 3.3 shows the summary of simulation results. From the table, it can be observed that the second-order methods result in lower training errors compared to the first-order algorithms. Figure 3.5 shows the average training error over epochs.

Though the training errors of mBFGS and mNAQ are comparable, mNAQ converges much faster compared to mBFGS and mNAQ $\mu = 0.8$ performs the best. However, it is sometimes the case where inspite of the training and test erros being low, the trained neural network predicted output may not be in close approximation to the actual expected response. Thus, we verify the goodness of the mNAQ

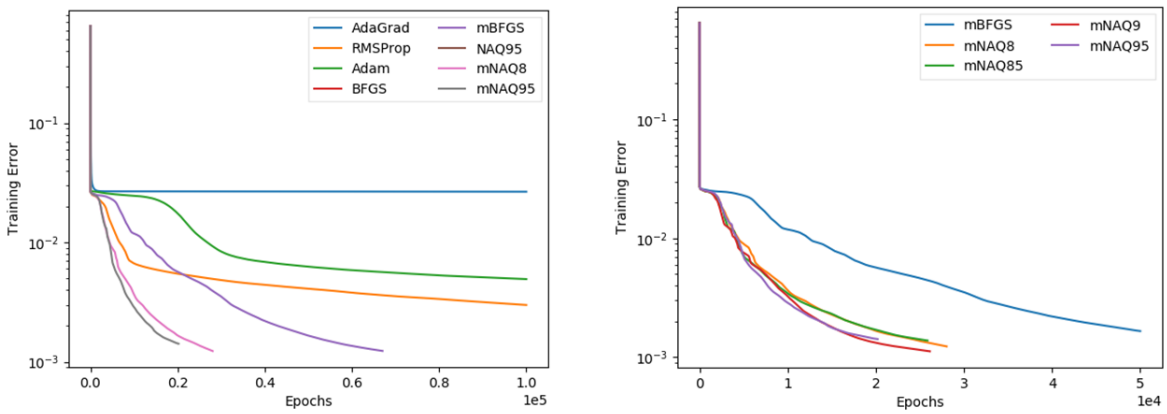


Figure 3.5: Average training error vs epochs over 15 trials for LPF.

trained neural network model by comparing its output to the true response on a test dataset. Figure 3.6 illustrates the output of the trained neural network with mNAQ $\mu = 0.8$ for lengths $d = 13$mm, 17mm, 15mm and 19mm. The output of the trained model is close to the original test dataset. Thus, we can conclude that the neural network predicted models can be used effectively in practical models.

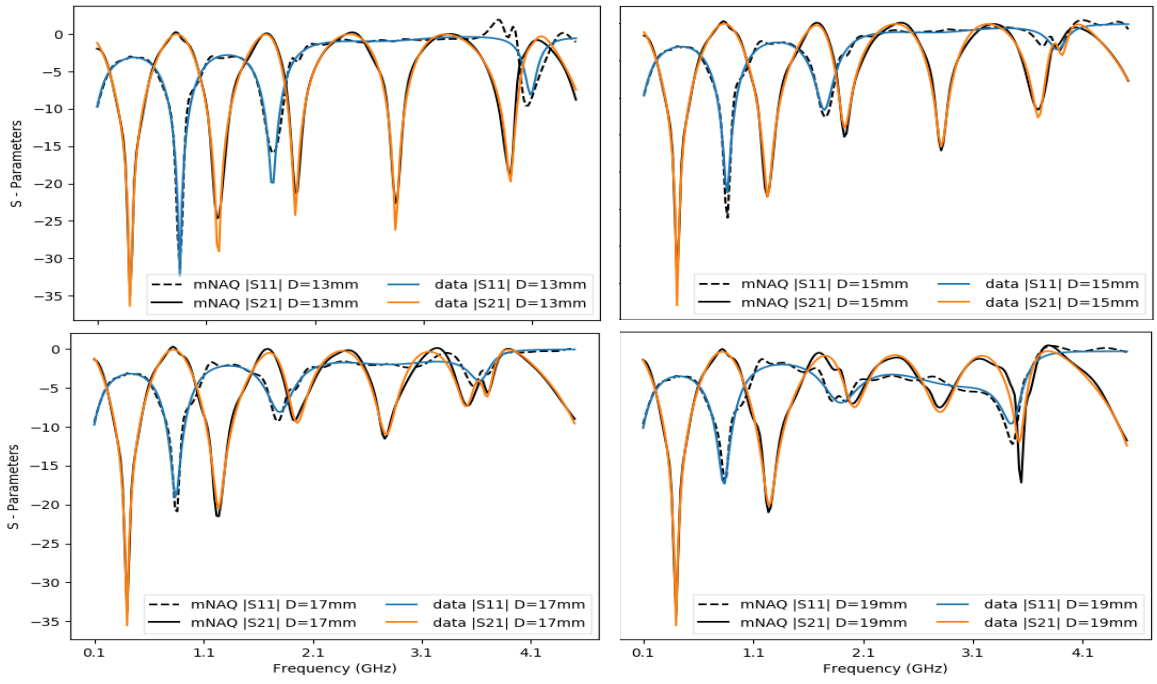


Figure 3.6: The comparison of network models of mNAQ *vs* original test data of LPF.

3.4.4 Op-Amp circuit design optimization problem

Operational amplifiers are one of the most commonly used circuits. However, determining the MOSFET channel width and length is a tedious process and usually determined by the designer's experience followed by repetitive tuning, simulations and redesigning. In this example, we consider a two-stage, Miller compensated op-amp [35] and present the results of determining the transistor sizing using

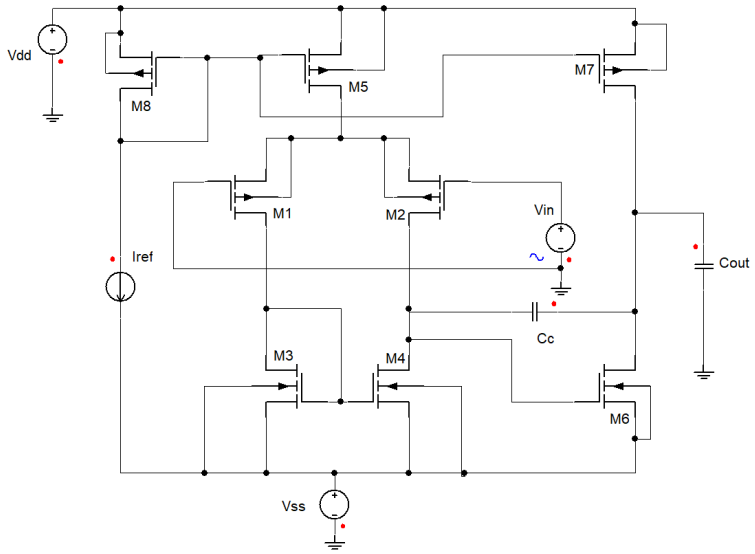


Figure 3.7: Two-Stage Op-Amp Schematic

neural networks [29]. The schematic of the op-amp under consideration is shown in Figure 3.8. The circuit consists of eight MOS transistors M1 to M8. M8 and M5 form a current mirror to supply the input differential pair M1-M2 with bias current I_{ref}. M1-M2 is actively loaded with the current mirror formed by M3-M4. The capacitor C_c is used for frequency compensation. For the generation of

Table 3.4: Design Specification

Parameter	Value
Supply Voltage	$\pm 2.5V$
$\mu_n C_{ox}$	$160\mu A/V^2$
$\mu_p C_{ox}$	$40\mu A/V^2$
Unity GBW	$>$ 1 MHz
Open Loop Gain A_o(dB)	>50 dB
Phase Margin	>60 deg

training and test dataset, a similar strategy to [36] was adopted. Channel length (L) was fixed to $1\mu m$ to avoid short channel effects. Based on the desired specifications shown in Table 3.4, an initial set of channel width (W) were chosen. 5% of the width in each case was varied with a uniform distribution i.e with 10 points in the range $[0.95W_n, 1.05W_n]$. Further, we set $W_1 = W_2$ and $W_3 = W_4$ to enforce symmetry. I_{ref} was varied in the range {60 μA - 100 μA} and load capacitance C_L = {2pF, 10pF}. Using standard design equations, the parameters DC open loop gain (A_o) in dB, unity gain bandwidth (GBW), slew rate (SR), phase margin (PM) and power dissipation (P_{diss}) were calculated. The neural network structure used was 7-4-18-10-9 with sigmoid activation functions. The input to the neural network comprised of {A_o, GBW, SR, PM, P_{diss}, I_{ref}, C_L}. 347 samples normalized in the range [0,1] were generated using a similar strategy to [36]. The terminate condition was set to 10^{-2}. The neural network output comprised of $\{W_1, W_2, W_3, W_4, W_5, W_6, W_7, W_8, C_c\}$. As seen from the previous EDA example, the choice of the momentum term μ is a hyperparameter that requires tuning. Hence we evaluate the performance with the adaptive momentum scheme as discussed in the example in section

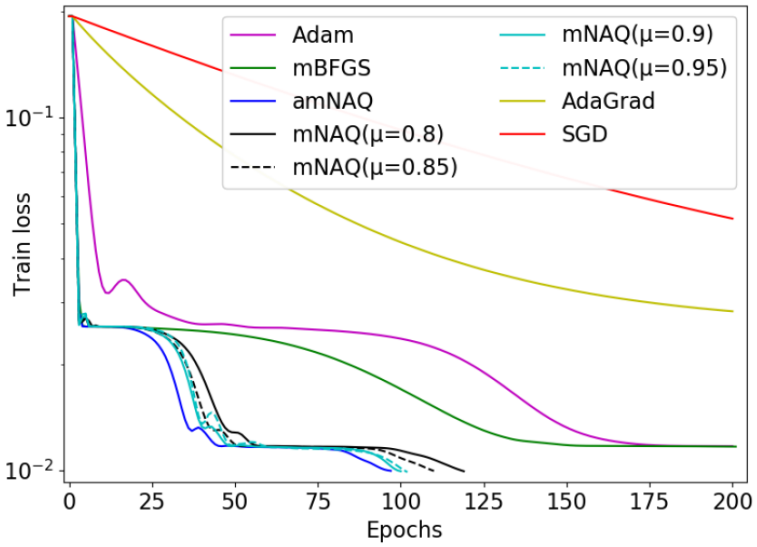


Figure 3.8: Train loss over 200 epochs (best case)

3.4.2. Figure 3.8 shows the training loss over 200 epochs for the best case trial. Table 3.5 shows the average, best and worst case train and test errors over 30 independent trials. Further the convergence rate (CR) and the average number of epochs required for convergence is also tabulated in Table 3.5. From the table, it can be observed that the performance of mNAQ with fixed values varies for different values of the momentum term, thus confirming the need for an adaptive momentum scheme. However, it is evident from Figure 3.8 that the second-order methods have faster convergence compared to the first-order methods. The result also confirms that the adaptive momentum amNAQ method converges to the desired value much faster compared to the other algorithms with a good convergence rate and low test error. From the results obtained, it can be confirmed that the second-order amNAQ can be efficiently used in solving transistor sizing problems.

Table 3.5: Summary of the results over 30 trials

	Algorithm	$E_{tr}(\mathbf{w})(\times 10^{-3})$ Ave/Best/Worst	CR (%)	Avg epochs	$E_{te}(\mathbf{w})(\times 10^{-3})$ Ave/Best/Worst
	SGD	66.4/43.2/113.3	-	200	68.4/45.9/118.6
	AdaGrad	35.1/26.9/53.7	-	200	36.8/29.5/57.5
	Adam	11.8/11.3/16.4	-	200	13.6/13.1/17.9
	BFGS	11.4/11.3/11.5	-	200	13.2/13.1/13.3
mNAQ	$\mu = 0.8$	10.0/9.9/11.2	90	161	11.9/11.6/13.0
mNAQ	$\mu = 0.85$	10.0/9.9/11.1	93.3	156	11.9/11.6/12.9
mNAQ	$\mu = 0.9$	9.9/9.9/10.5	93.3	156	11.8/11.6/12.4
mNAQ	$\mu = 0.95$	10.3/9.9/11.3	63.3	178	12.1/11.5/13.2
	amNAQ	9.9/9.9/11.3	96.7	146	11.8/11.6/13.1

4

Accelerated Stochastic Quasi-Newton Methods

Neural networks have shown to be effective in innumerous real-world applications. There are several applications that require large neural network models with massive amounts of training data to achieve good accuracies and low errors. It is expected that the neural network training imposes relatively lower computational and memory demands, in which case a full-batch approach (as in Chapter 3) is not suitable. Thus, in such large scale optimization problems, a stochastic / mini-batch approach is more desirable. Thus in this chapter we introduce stochastic extentions of the Nesterov's accelerated and momentum accelerated quasi-Newton method. This chapter is based on the works published in [37, 38].

4.1 Introduction

With the growing demand in machine learning for large scale applications, it is quite common to encounter optimization problems with millions of training examples and millions of parameters to train upon. In order to cope with the high demands of computation resources and time imposed by such models, several stochastic algorithms have been developed. Stochastic algorithms use a small subset of data (mini-batch) in the training at each iteration. These methods are particularly of relevance in examples of a continuous stream of data, where the partial data is to be modeled as it arrives. Since the stochastic or online methods operate on small subsamples of the data and its gradients, they significantly reduce the computational and memory requirements. However, conventional optimization methods are not well equipped to deal with the challenges that arise in large scale stochastic optimization. The non-linearity, scale and stochasticity inherent in the neural network models give rise to highly complex optimization problems and this has stimulated a wide stream of algorithmic research.

4.2 Background

Several works have been devoted to stochastic first-order methods such as stochastic gradient descent (SGD) [10, 11] and its variance-reduced forms [5, 9, 39], AdaGrad [12], RMSprop [21] and Adam [22]. First order methods are popular due to its simplicity and optimal complexity. However, incorporating the second-order curvature information have shown to improve convergence. But one of the major drawbacks in second-order methods is its need for high computational and memory resources. Thus several approximations have been proposed under Newton [40, 41] and quasi-Newton [42] methods in order to make use of the second-order information while keeping the computational load minimal.

Unlike the first-order methods, getting quasi-Newton methods to work in a stochastic setting is challenging and has been an active area of research [43]. The oBFGS method [44] is one of the early stable stochastic quasi-Newton methods, in which the gradients are computed twice using the same sub-sample, to ensure stability and scalability. Recently there has been a surge of interest in designing efficient stochastic second-order variants which are better suited for large scale problems. [45] proposed a regularized stochastic BFGS method (RES) that modifies the proximity condition of BFGS. [46] further analyzed the global convergence properties of stochastic BFGS and proposed an online L-BFGS method. [47] proposed a stochastic limited memory BFGS (SQN) through sub-sampled Hessian vector products. [48] proposed a general framework for stochastic quasi-Newton methods that assume noisy gradient information through first-order oracle (SFO) and extended it to a stochastic damped L-BFGS method (SdLBFGS). This was further modified in [49] by reinitializing the Hessian matrix at each iteration to improve convergence and normalizing the search direction to improve stability. There are also several other studies on stochastic quasi-Newton methods with variance reduction [7, 50, 51], sub-sampling [41, 52] and block updates [53]. Most of these methods have been proposed for solving convex optimization problems, but training of neural networks for non-convex problems have not been mentioned in their scopes. The focus of this chapter is on training neural networks for non-convex problems with methods similar to that of the oBFGS in [44] and RES [45, 46], as they are stochastic extensions of the classical quasi-Newton method. Thus, tht other sophisticated algorithms such as those in [7, 41, 47–53] are excluded from comparision.

Stochastic BFGS Method (oBFGS)

The online BFGS method proposed by Schraudolph et al in [44] is a fast and scalable stochastic quasi-Newton method suitable for convex functions. The changes proposed to the BFGS method in [44] to work well in a stochastic setting are discussed as follows. Usually line search methods for determing the step size is not suitable for stochastic training. Thus the line search is replaced with a decay schedule such as

$$\alpha_k = \tau/(\tau + k) \cdot \alpha_0, \tag{4.1}$$

where $\alpha_0, \tau > 0$ provided the Hessian matrix is positive definite, thus restricting to convex optimization problems. Since line search is eliminated, the first parameter update is scaled by a small value. Further, to improve the performance of oBFGS, the step size is divided by an analytically determined constant c. An important modification is the computation of $\mathbf{y}_k$, the difference of the last two gradients is computed on the same sub-sample X_k [44, 45] as given below,

$$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}, X_k) - \nabla E(\mathbf{w}_k, X_k). \tag{4.2}$$

This however doubles the cost of gradient computation per iteration but is shown to outperform natural gradient descent for all batch sizes [44]. The oBFGS algorithm is as shown in Algorithm 4.1.

Stochastic Limited Memory BFGS (oLBFGS)

[44] further extends the oBFGS method to limited memory form by determining the search direction $\mathbf{g}_k$ using the two-loop recursion (Appendix A.1) and storing the curvature information pair instead of the update in step 11 of Algorithm 4.1. The Hessian update is omitted and instead the last m curvature

Algorithm 4.1 Stochastic BFGS Method - oBFGS

Require: minibatch X_k, k_{max} and $\lambda \geq 0$,
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$, $\mathbf{H}_k = \epsilon\mathbf{I}$ and $\mathbf{v}_k = 0$
1: $k \leftarrow 1$
2: **while** $k < k_{max}$ **do**
3: $\nabla\mathbf{E}_1 \leftarrow \nabla E(\mathbf{w}_k, X_k)$
4: $\mathbf{g}_k \leftarrow -\mathbf{H}_k^{\text{BFGS}} \nabla E(\mathbf{w}_k, X_k)$
5: Determine α_k using (4.1)
6: $\mathbf{v}_{k+1} \leftarrow \alpha_k \mathbf{g}_k$
7: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
8: $\nabla\mathbf{E}_2 \leftarrow \nabla E(\mathbf{w}_{k+1}, X_k)$
9: $\mathbf{s}_k \leftarrow \mathbf{w}_{k+1} - \mathbf{w}_k$
10: $\mathbf{y}_k \leftarrow \nabla\mathbf{E}_2 - \nabla\mathbf{E}_1 + \lambda\mathbf{s}_k$
11: Update $\mathbf{H}_k^{\text{BFGS}}$ using (3.26)
12: $k \leftarrow k + 1$
13: **end while**

pairs $\mathbf{s}_k$ and $\mathbf{y}_k$ are stored. This brings down the computation complexity to $2bd + 6md$ where b is the batch size, d is the number of parameters, and m is the memory size. To improve the performance by averaging sampling noise, step 7 of Algorithm A.1 is replaced by (4.3) where σ_k and γ_k correspond to the curvature information pair $\mathbf{s}_k$ and $\mathbf{y}_k$.

$$\eta_k = \begin{cases} \epsilon\eta_k & \text{if } k = 1, \\ \dfrac{\eta_k}{\min(\mathrm{k}, \mathrm{m})} \displaystyle\sum_{i=1}^{\min(\mathrm{k},\mathrm{m})} \frac{\sigma_{k-i}^{\mathrm{T}} \gamma_{k-i}}{\gamma_{k-i}^{\mathrm{T}} \gamma_{k-i}} & \text{otherwise.} \end{cases} \tag{4.3}$$

4.3 Stochastic BFGS with Nesterov's Acceleration

4.3.1 Stochastic NAQ Method

The oBFGS method proposed in [44] computes the gradient of a sub-sample minibatch X_k twice in one iteration. This is comparable with the inherent nature of NAQ which also computes the gradient twice in one iteration. Thus by applying suitable modifications to the original NAQ algorithm, we achieve a stochastic version of the Nesterov's Accelerated Quasi-Newton method. The proposed modifications for a stochastic NAQ method is discussed below in its full and limited memory forms [37]. The NAQ algorithm computes two gradients, $\nabla E(\mathbf{w}_k + \mu\mathbf{v}_k)$ and $\nabla E(\mathbf{w}_{k+1})$ to calculate $\mathbf{y}_k$ as shown in (3.36). On the other hand, the oBFGS method proposed in [44] computes the gradient $\nabla E(\mathbf{w}_k, X_k)$ and $\nabla E(\mathbf{w}_{k+1}, X_k)$ to calculate $\mathbf{y}_k$ as shown in (4.2). Therefore, oNAQ can be realised to calculate $\nabla E(\mathbf{w}_k + \mu\mathbf{v}_k, X_k)$ and $\nabla E(\mathbf{w}_{k+1}, X_k)$ as shown in Algorithm 4.2. Thus in oNAQ, the $\mathbf{y}_k$ vector is given by (4.4) where $\lambda\mathbf{s}_k$ is used to guarantee numerical stability [27, 54, 55].

$$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}, X_k) - \nabla E(\mathbf{w}_k + \mu\mathbf{v}_k, X_k) + \lambda\mathbf{s}_k, \tag{4.4}$$

Further, unlike in full batch methods, the updates in stochastic methods have high variance resulting in the objective function to fluctuate heavily. This is due to the updates being performed based on small sub-samples of data. This can be seen more prominently in case of the limited memory version where

Algorithm 4.2 Stochastic NAQ Method - oNAQ

Require: minibatch X_k, $0 < \mu < 1$ and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$, $\mathbf{H}_k^{\text{NAQ}} = \epsilon\mathbf{I}$ and $\mathbf{v}_k = 0$

$k \leftarrow 1$
while $k < k_{max}$ **do**
  $\nabla\mathbf{E}_1 \leftarrow \nabla E(\mathbf{w}_k + \mu\mathbf{v}_k, X_k)$
  $\hat{\mathbf{g}}_k \leftarrow -\mathbf{H}_k^{\text{NAQ}} \nabla E(\mathbf{w}_k + \mu\mathbf{v}_k, X_k)$
  $\hat{\mathbf{g}}_k = \hat{\mathbf{g}}_k / \|\hat{\mathbf{g}}_k\|_2$
  Determine α_k using (4.6)
  $\mathbf{v}_{k+1} \leftarrow \mu\mathbf{v}_k + \alpha_k\hat{\mathbf{g}}_k$
  $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
  $\nabla\mathbf{E}_2 \leftarrow \nabla E(\mathbf{w}_{k+1}, X_k)$
  $\mathbf{s}_k \leftarrow \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu\mathbf{v}_k)$
  $\mathbf{y}_k \leftarrow \nabla\mathbf{E}_2 - \nabla\mathbf{E}_1 + \lambda\mathbf{s}_k$
  Update $\mathbf{H}_k^{\text{NAQ}}$ using (3.36)
  $k \leftarrow k + 1$
end while

the updates are based only on m recent curvature pairs. Thus in order to improve the stability of the algorithm, we introduce direction normalization as

$$\hat{\mathbf{g}}_k = \frac{\hat{\mathbf{g}}_\mathbf{k}}{\|\hat{\mathbf{g}}_k\|_2}, \tag{4.5}$$

where $\|\hat{\mathbf{g}}_k\|_2$ is the l_2 norm of the search direction $\hat{\mathbf{g}}_k$. Normalizing the search direction at each iteration ensures that the algorithm does not move too far away from the current objective [49]. Figure 4.1 illustrates the effect of direction normalization on oBFGS and the proposed oNAQ method. The solid lines indicate the moving average. As seen from the figure, direction normalization improves the performance of both oBFGS and oNAQ. Therefore, in this study we include direction normalization for oBFGS also.

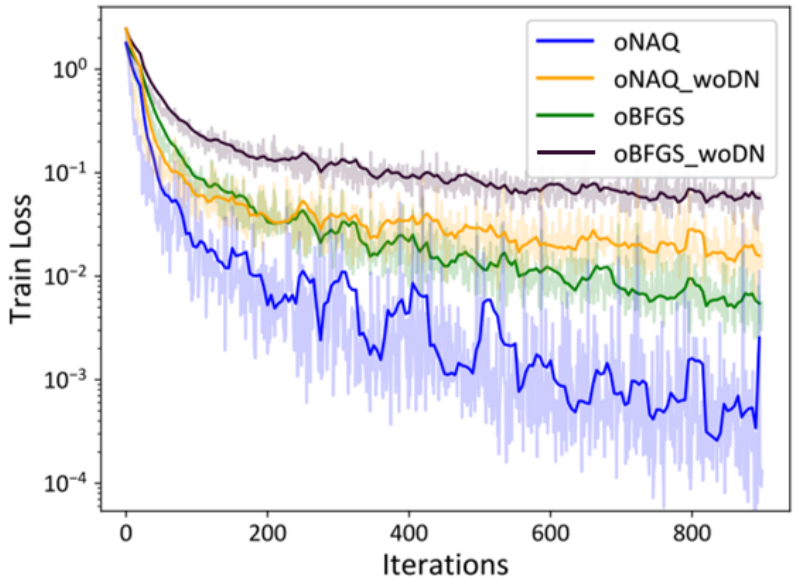


Figure 4.1: Effect of direction normalization on 8x8 MNIST with b = 64 and $\mu = 0.8$.

The next proposed modification is with respect to the step size. In full batch methods, the step size or the learning rate is usually determined by line search methods satisfying either Armijo or Wolfe conditions. However, in stochastic methods, line searches are not quite effective since search conditions apply global validity. This cannot be assumed when using small local sub-samples [44]. Several studies show that line search methods does not necessarily ensure global convergence and have proposed methods that eliminate line search [27, 54, 55]. Moreover, determining step size using line search methods involves additional function computations until the search conditions such as the Armijo or Wolfe condition is satisfied. Hence we determine the step size using a simple learning rate schedule. Common learning rate schedules are polynomial decays and exponential decay functions. In this study, we determine the step size using a polynomial decay schedule [56]

$$\alpha_k = \frac{\alpha_0}{\sqrt{k}}, \tag{4.6}$$

where α_0 is usually set to 1. If the step size is too large, which is the case in the initial iterations, the learning can become unstable. This is stabilized by direction normalization. A comparison of common learning rate schedules are illustrated in Figure 4.2

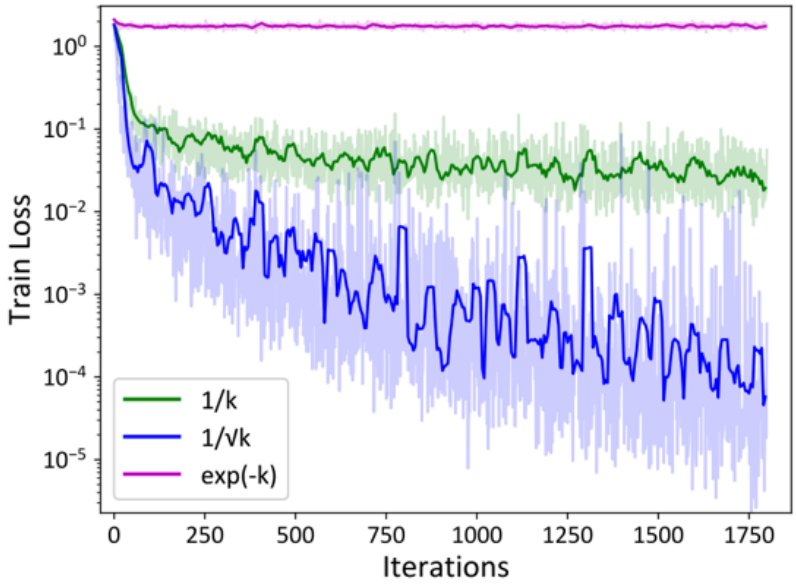


Figure 4.2: Comparison of α_k schedules on 8x8 MNIST with b = 64 and $\mu = 0.8$.

The proposed stochastic NAQ algorithm is shown in Algorithm 4.2. Note that the gradient is computed twice in one iteration, thus making the computational cost same as that of the stochastic BFGS (oBFGS) proposed in [44].

4.3.2 Stochastic Limited-Memory NAQ (oLNAQ)

Stochastic LNAQ can be realized by making modifications to Algorithm 4.2 similar to LNAQ. The search direction $\hat{\mathbf{g}}_k$ in step 4 is determined by Algorithm A.1. oLNAQ like LNAQ uses the last m curvature pairs $\{\mathbf{s}_k, \mathbf{y}_k\}$ to estimate the Hessian matrix instead of storing and computing on a dxd matrix. Therefore, the implementation of oLNAQ does not require initializing or updating the Hessian matrix. Hence step 12 of Algorithm 4.2 is replaced by storing the last m curvature pairs $\{\mathbf{s}_k, \mathbf{y}_k\}$.

Finally, in order to average out the sampling noise in the last m steps, we replace step 7 of Algorithm A.1 by (4.3) where σ_k is $\mathbf{s}_k$ and γ_k is $\mathbf{y}_k$. Note that an additional $2md$ evaluations are required to compute (4.3). However the overall computation cost of oLNAQ is much lesser than that of oNAQ and the same as oLBFGS.

4.3.3 Simulation Results

The performance of the proposed stochastic methods oNAQ and oLNAQ is evaluated on classification and regression problems. The performance of the classification task is evaluated on a multi-layer neural network (MLNN) and a simple convolution neural network (CNN). The algorithms oNAQ, oBFGS, oLNAQ and oLBFGS are implemented in Tensorflow using the ScipyOptimizerInterface class. For a fair CPU time comparison, we implement the Adam optimzer also using the ScipyOptimizerInterface class.

Classification Problem

The 28×28 MNIST dataset [57] is used to illustrate the performance of the proposed algorithm on the classification task. It was observed that oNAQ and oLNAQ required fewer epochs compared to oBFGS, oLBFGS, Adam and SGD. In terms of computation time, o(L)BFGS and o(L)NAQ required longer time compared to the first-order methods. This is due to the Hessian computation and two gradient calculations per iteration. Further, the oBFGS and oNAQ per iteration time difference compared to first-order methods is much larger than that of the limited memory algorithms with memory $m = 4$. It was observed that for the same time, the second-order methods perform significantly better compared to the first-order methods, thus confirming that the extra time taken by the second-order methods does not adversely affect its performance. Thus, the train loss and test accuracy versus time is used to evaluate the performance of the proposed method.

Results on Multi-Layer Neural Networks

A simple MLNN with two hidden layers is considered. ReLU activation function and softmax cross-entropy loss function is used. Each layer except the output layer is batch normalized. Due to large number of parameters, the performace of only the limited memory methods are illustrated below. Figure 4.3 shows the results of oLNAQ on the 28×28 MNIST dataset for batch size $b = 64$ and $b = 128$. The results indicate that oLNAQ clearly outperforms oLBFGS and SGD for even small batch sizes. On comparing with Adam, oLNAQ is in close competition with Adam for small batch sizes such as $b = 64$ and performs better for larger batch sizes such as $b = 128$.

Results on Convolution Neural Network

The performance of the proposed algorithm is evaluated on a simple convolution neural network (CNN) with two convolution layers followed by a fully connected layer. Sigmoid activation functions and softmax cross-entropy error function are used. The CNN architecture comprises of two convolution layers of 3 and 5 5x5 filters respectively, each followed by 2x2 max pooling layer with stride 2. The convolution layers are followed by a fully connected layer with 100 hidden neurons. The batch size $b = 128$ and limited mem $m = 4$ is chosen with a parameters size of $d = 260,068$. Figure 4.4 shows

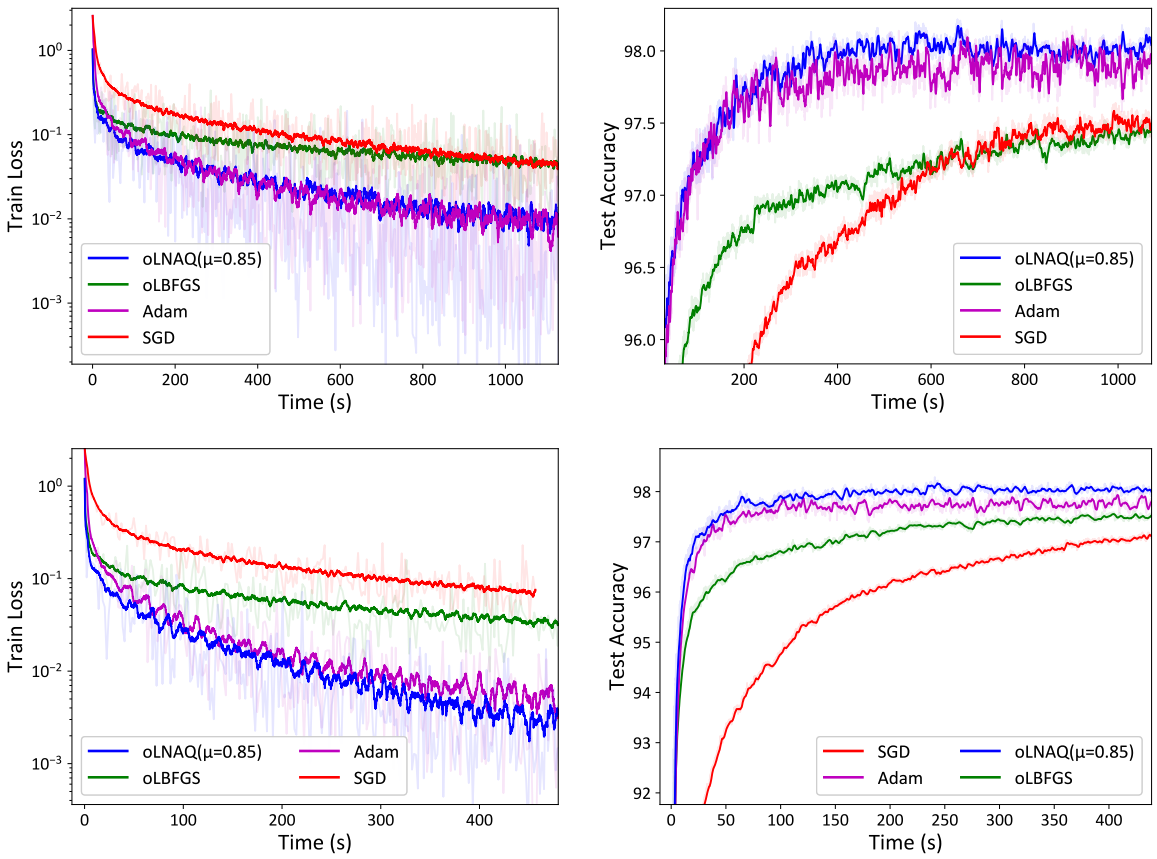


Figure 4.3: Results on 28×28 MNIST for $b = 64$ (top) and $b = 128$ (bottom).

the results of oLNAQ on the simple CNN. The CNN results show similar performance as that of the results on multi-layer neural network where oLNAQ outperforms SGD and oBFGS. Comparing with Adam, oLNAQ is much faster in the first few epochs and becomes closely competitive to Adam as the number of epochs increases. Calculation of the gradient twice per iteration increases the time per iteration when compared to the first-order methods. However this is compensated well since the overall performance of the algorithm is much better compared to Adam and SGD. Also the number of epochs required to converge to low error and high accuracies is much lesser than the other algorithms. In other words, the same accuracy or error can be achieved with lesser amount of training data.

Regression Problem

The performance of the proposed stochastic methods on regression problems is illustrated. For this task, the benchmark white wine quality prediction task [58] is chosen and evaluated on a multi-layer neural network with 2 hidden layers. The task is to predict the quality of the white wine on a scale of 3 to 9 based on 11 physiochemical test values. Sigmoid activation function and mean squared error (MSE) function is used. Each layer except the output layer is batch normalized. The dataset is z-normalized to have zero mean and unit variance. The dataset is split in 80-20 % for train and test set. For the regression problem, oNAQ with smaller values of momemtum $\mu = 0.8$ and $\mu = 0.85$ show similar performance as that of oBFGS. Larger values of momentum resulted in better performance. Hence a value of $\mu = 0.95$ is chosen as it showed faster convergence compared to the other methods. Further

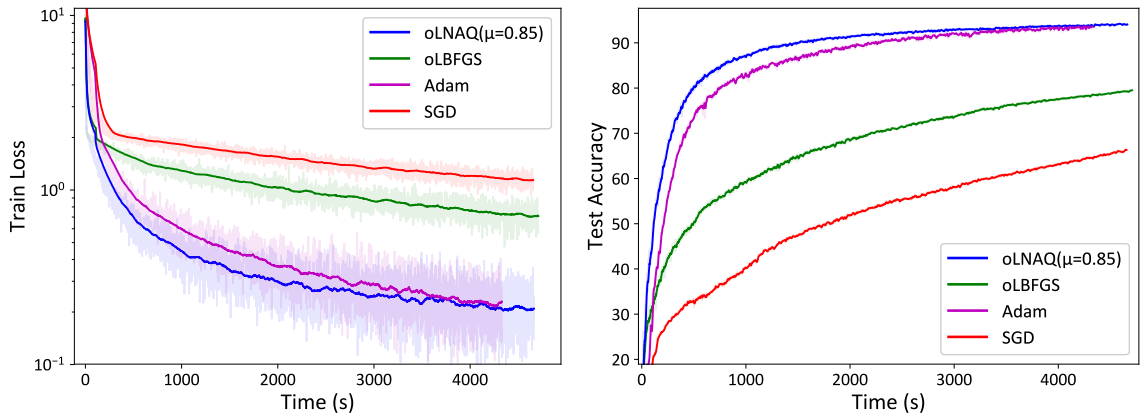


Figure 4.4: CNN Results on 28×28 MNIST with $b = 128$.

comparing the performance for different batch sizes, it was observed that for smaller batch sizes such as $b = 32$, oNAQ is close in performance with Adam and oLNAQ is initially fast and gradually becomes close to Adam. For bigger batch sizes such as $b = 64$, oNAQ and oLNAQ are faster in convergence initially. Over time, oLNAQ continues to result in lower error while oNAQ gradually becomes close to Adam. Figure 4.5 shows the root mean squared error (RMSE) versus time for batch sizes $b = 32$ and $b = 64$.

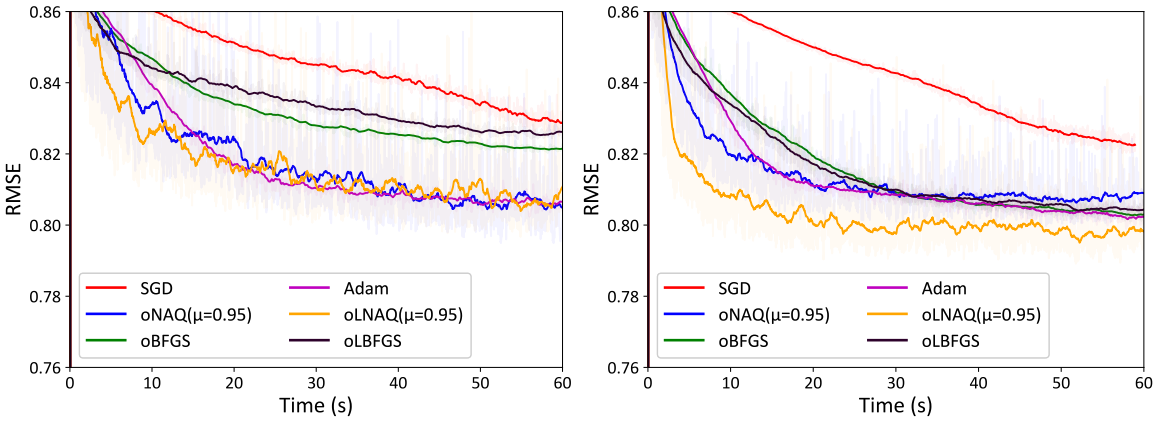


Figure 4.5: Results of Wine Quality Dataset for $b = 32$ (left) and $b = 64$ (right).

4.4 Stochastic BFGS with Momentum Acceleration

4.4.1 Stochastic MoQ Method

Both o(L)BFGS and o(L)NAQ methods compute the gradient twice per iteration. However from the results presented above, it was clear that the performance in terms of computaion time was not adversely affected. However, the momentum accelerated quasi-Newton method (MoQ) [33] showed that the Nesterov's accelerated gradient in NAQ can be approximated as a linear combination of past gradients as shown below.

$$\nabla \mathbf{E}(\mathbf{w}_k + \mu \mathbf{v}_k) \approx (1 + \mu)\nabla \mathbf{E}(\mathbf{w}_k) - \mu \nabla \mathbf{E}(\mathbf{w}_{k-1}) \tag{4.7}$$

Extending this approximation to o(L)NAQ, we further propose a stochastic momentum accelerated quasi-Newton (oMoQ) method. The algorithm is as shown in Algorithm 4.3. The Hessian $\mathbf{H}_k^{\text{MoQ}}$ is updated by as

$$\mathbf{H}_{k+1}^{\text{MoQ}} = (\mathbf{I} - \rho_k \mathbf{s}_k \mathbf{y}_k^{\text{T}}) \mathbf{H}_k^{\text{MoQ}} (\mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^{\text{T}}) + \rho_k \mathbf{s}_k \mathbf{s}_k^{\text{T}}, \tag{4.8}$$

where

$$\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu \mathbf{v}_k) = \mathbf{w}_{k+1} - (1+\mu)\mathbf{w}_k + \mu \mathbf{w}_{k-1}, \tag{4.9}$$

$$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}, \mathbf{X}_{k+1}) - (1+\mu)\nabla E(\mathbf{w}_k, \mathbf{X}_k) + \mu \nabla E(\mathbf{w}_{k-1}, \mathbf{X}_{k-1}). \tag{4.10}$$

Note that unlike o(L)BFGS and o(L)NAQ, the proposed method computes only one gradient per iteration, while the gradient computed with respect to the previous batch $\mathbf{X}_k$ is stored in memory. The computation cost is thus reduced by bd, where b is the batch size and d is the number of parameters. From (4.10), we can observe that the curvature information term $\mathbf{y}_k$ is a computed based on three mini-batch samples, which could result in increased stochastic sampling noise. Limited memory oLMoQ is formulated by computing the search direction $\mathbf{g}_k$ (step 5) using the two-loop recursion (Appendix A.1).

Algorithm 4.3 Stochastic MoQ Method - oMoQ

Require: learning rate schedule, $0 < \mu < 1$ and k_{max}
Initialize: $\mathbf{w}_k \in \mathbb{R}^d$, $\mathbf{H}_k = \epsilon \mathbf{I}$ and $\mathbf{v}_k = 0$
1: Calculate $\nabla \mathbf{E}(\mathbf{w}_k, X_k)$
2: **while** $\|\nabla \mathbf{E}(\mathbf{w}_k)\| > \epsilon$ and $k < k_{\max}$ **do**
3: Determine learning rate α_k
4: $\nabla \mathbf{E}_1 = (1+\mu)\nabla \mathbf{E}(\mathbf{w}_k, X_k) - \mu \nabla \mathbf{E}(\mathbf{w}_{k-1}, X_{k-1})$
5: $\mathbf{g}_k \leftarrow -\mathbf{H}_k^{\text{MoQ}} \nabla \mathbf{E}_1$
6: $\mathbf{g}_k = \mathbf{g}_k / \|\mathbf{g}_k\|_2$
7: $\mathbf{v}_{k+1} \leftarrow \mu \mathbf{v}_k + \alpha_k \mathbf{g}_k$
8: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
9: Store $\nabla \mathbf{E}(\mathbf{w}_k, X_k)$
10: Select mini-batch X_{k+1}
11: Calculate $\nabla \mathbf{E}_2 = \nabla \mathbf{E}(\mathbf{w}_{k+1}, X_{k+1})$
12: $\mathbf{s}_k \leftarrow \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu \mathbf{v}_k)$
13: $\mathbf{y}_k \leftarrow \nabla \mathbf{E}_2 - \nabla \mathbf{E}_1 + \lambda \mathbf{s}_k$
14: Update $\mathbf{H}_k^{\text{MoQ}}$ using (25)
15: **end while**

4.4.2 MoQ Simulation Results

We illustrate the performance of the proposed stochastic methods oMoQ and oLMoQ the 28×28 MNIST dataset on a multi-layer neural network (MLNN), a simple convolution neural network (CNN) and LeNET-5 architectures. The algorithms oMoQ, oNAQ, oBFGS, oLMoQ, oLNAQ and oLBFGS are implemented in Tensorflow using the ScipyOptimizerInterface class.

Results on Feedforward Neural Networks

We consider a simple feedforward NN with two hidden layers. ReLU activation function and softmax cross-entropy loss function is used. Each layer except the output layer is batch normalized. We first

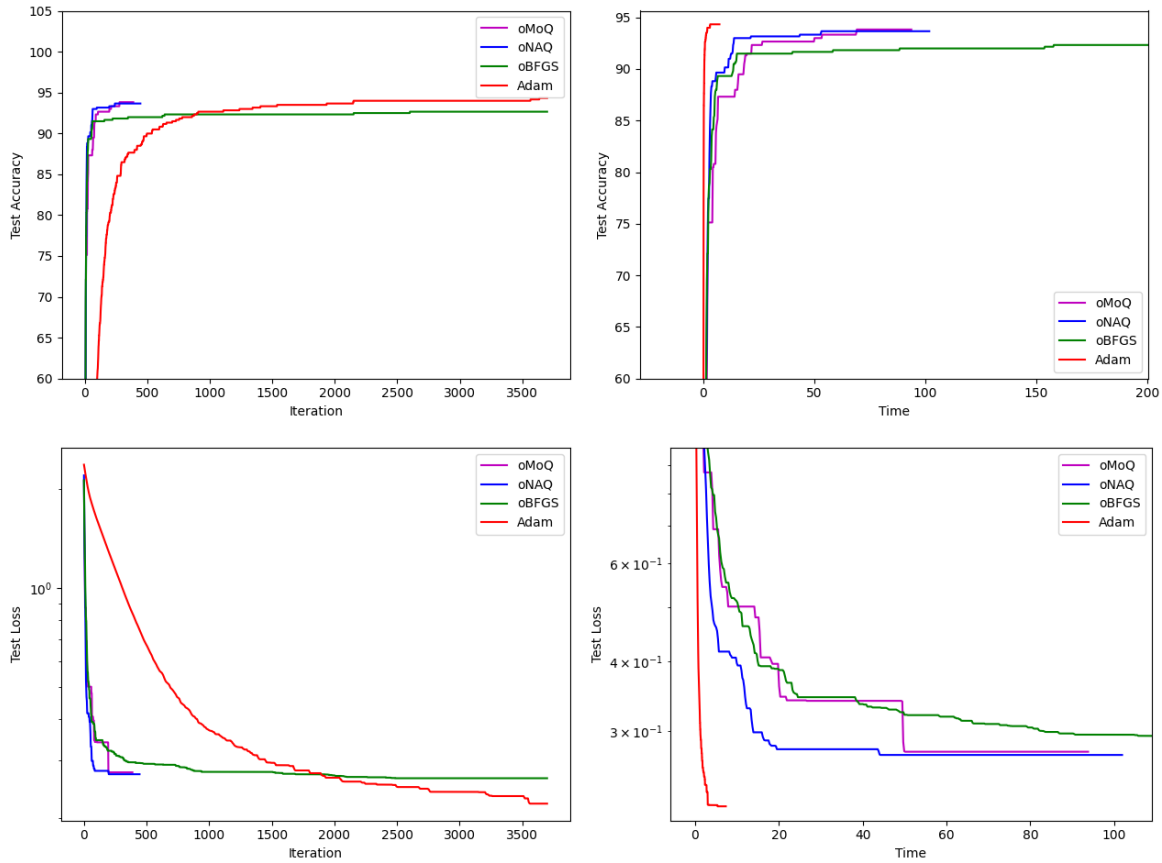


Figure 4.6: Feedforward NN results on 8×8 MNIST with $b = 32$.

evaluate the performance of the full memory oMoQ, oNAQ and oBFGS using the 8x8 reduced MNIST dataset [59]. The dataset comprises of 1797 samples which we divide in 75-25 % for training and testing. The NN architecture is $64 - 20 - 10 - 10$ and the number of parameter $d = 1620$. A batch size of 32 is chosen, $\mu = 0.8$ and $m = 10$. We terminate the training when the train loss is less than 10^{-3}. Figure 4.6 shows the results of the full memory algorithms. It was observed that oMoq terminated at 11 epochs, oNAQ at 13 epochs, oBFGS at 100 epochs and Adam at 217 epochs. As seen from the figure, the computation time of full memory methods are much larger than first-order methods like Adam. However the number of iterations required to reach the same level of accuracy is much more than that of oMoQ and oNAQ. Furthermore, for larger networks, the full memory scheme may not prove to have an advantage over the first-order methods unless implemented using parallel or distributed programming. To this end, limited memory methods prove to be more efficient. In the following sections, we evaluate the limited memory methods. However in this example, performance wise, we can observe that the first-order Adam is slow compared to second-order stochastic methods, and oNAQ and oMoQ exhibit better performance than oBFGS.

The number of parameter $d = 84,060$. The batch size is chosen to be 128, $\mu = 0.85$ and limited memory $m = 4$. Figure 4.7 shows the results on the feed forward neural network. From the results, it is clear that unlike the second-order methods, Adam requires more number of iterations to converge. oLMoQ performs better than oLBFGS as a result of momentum acceleration. Also, oLMoQ takes lesser time compared to oLBFGS and oLNAQ. Nevertheless, oLNAQ performs the best in this case.

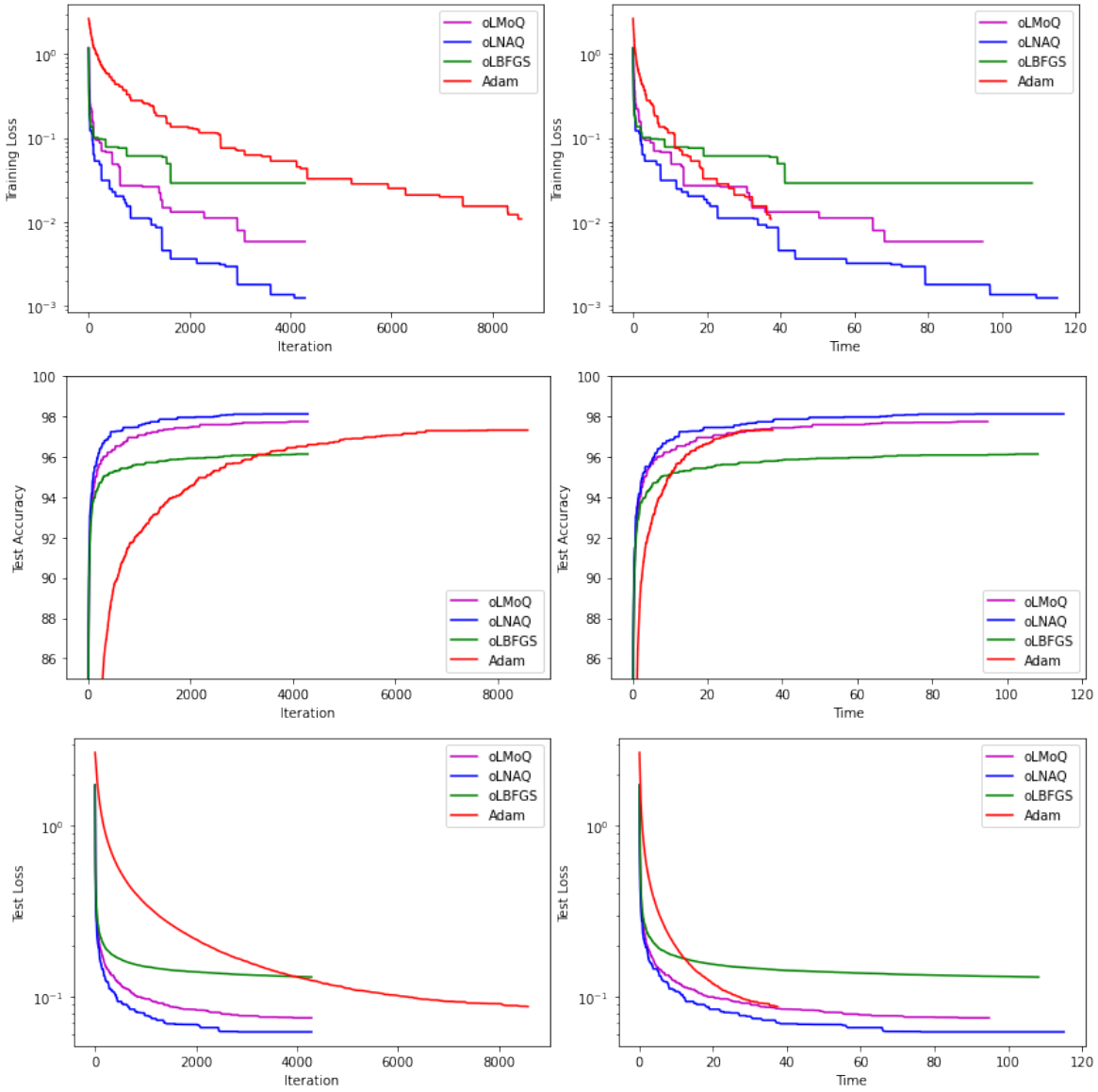


Figure 4.7: Feedforward NN results on 28×28 MNIST with $b = 128$.

Results on simple two layer Convolution Neural Network

We study the performance of the proposed algorithms on a simple convolution neural network (CNN) with two convolution layers followed by a fully connected layer. We use sigmoid activation functions and softmax cross-entropy error function. We evaluate the performances of oLNAQ and oLMoQ using the 28×28 MNIST dataset with a batch size of 128 and $\mu = 0.85$ and number of parameters $d = 26,068$. The CNN architecture comprises of two convolution layers of 3 and 5 5x5 filters respectively, each followed by 2x2 max pooling layer with stride 2. The convolution layers are followed by a fully connected layer with 100 hidden neurons. Figure 4.8 shows the results on the 2-layer CNN. Both iteration wise and time wise, oLMoQ performs better than Adam, oLBFGS and oLNAQ. As we can observe from the figure, the time taken to complete 20 epochs by Adam is much lesser than that of oLMoQ or oLNAQ. However, it also clear that even if we were to stop the training after 20s, the performance of oLMoQ is superior to that of Adam.

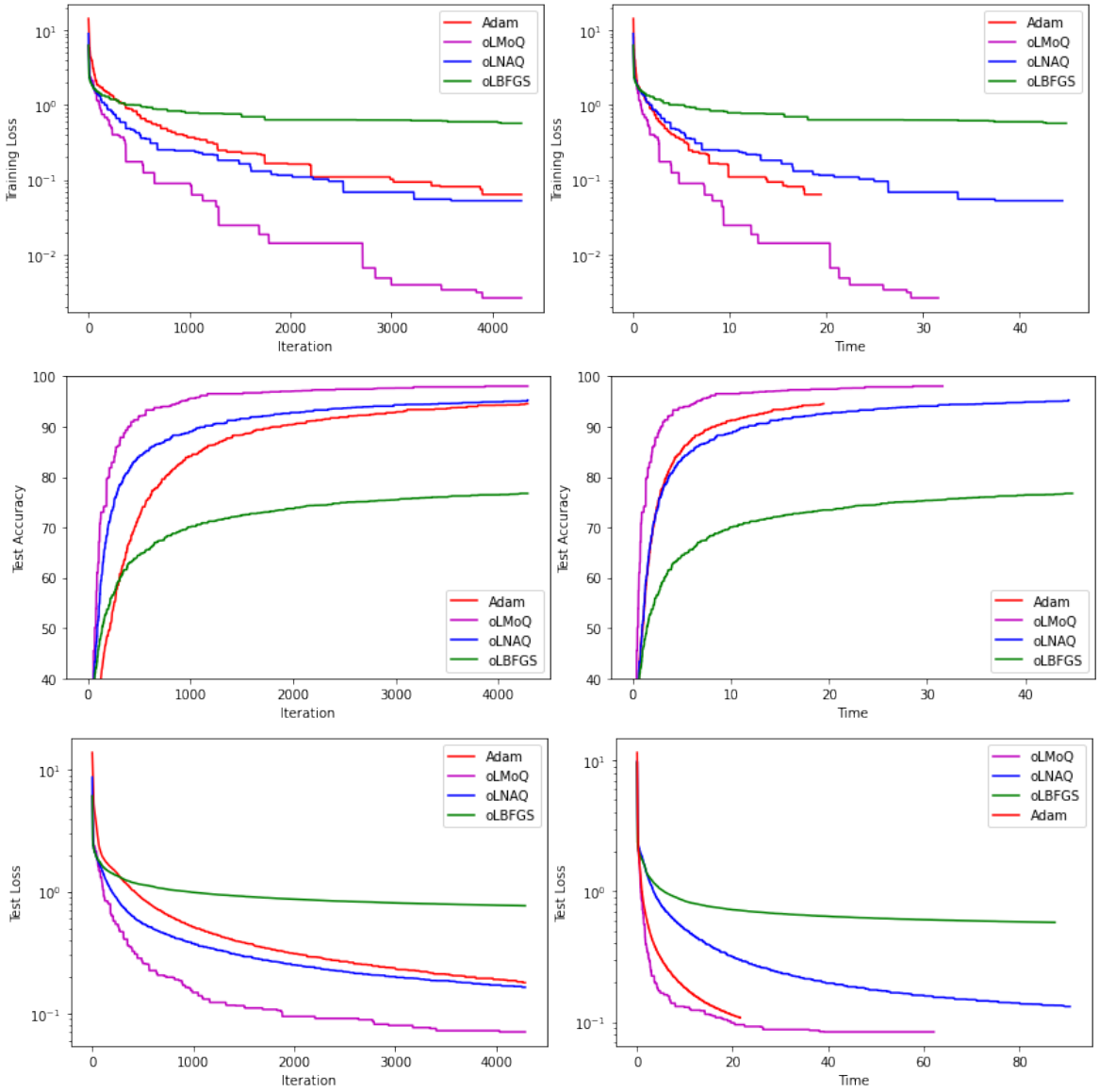


Figure 4.8: Simple 2 layer CNN results on 28×28 MNIST with $b = 128$.

Results on LeNET-5

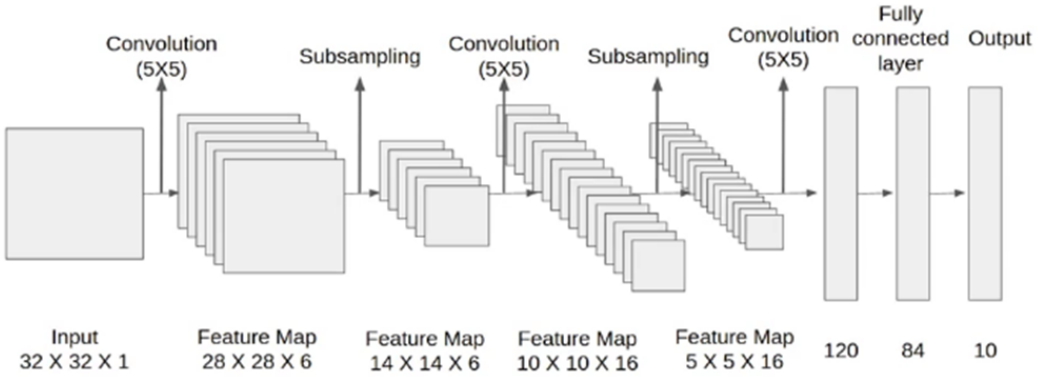


Figure 4.9: The LeNet-5 architecture (Source: Yann LeCun et al. [60])

The LeNet-5 architecture [60] consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier. The number of parameter $d = 61706$, batch size $b = 128$, $mu = 0.85$ and $m = 4$. Figure 4.10 shows the results of time versus test loss and test accuracy; and iteration versus test loss and test accuracy. From the figure, it is clear that the proposed oLMoQ outperforms oLBFGS, oLNAQ and Adam. It take relatively lesser time compared of oLBFGS and oLNAQ and is comparable to Adam.

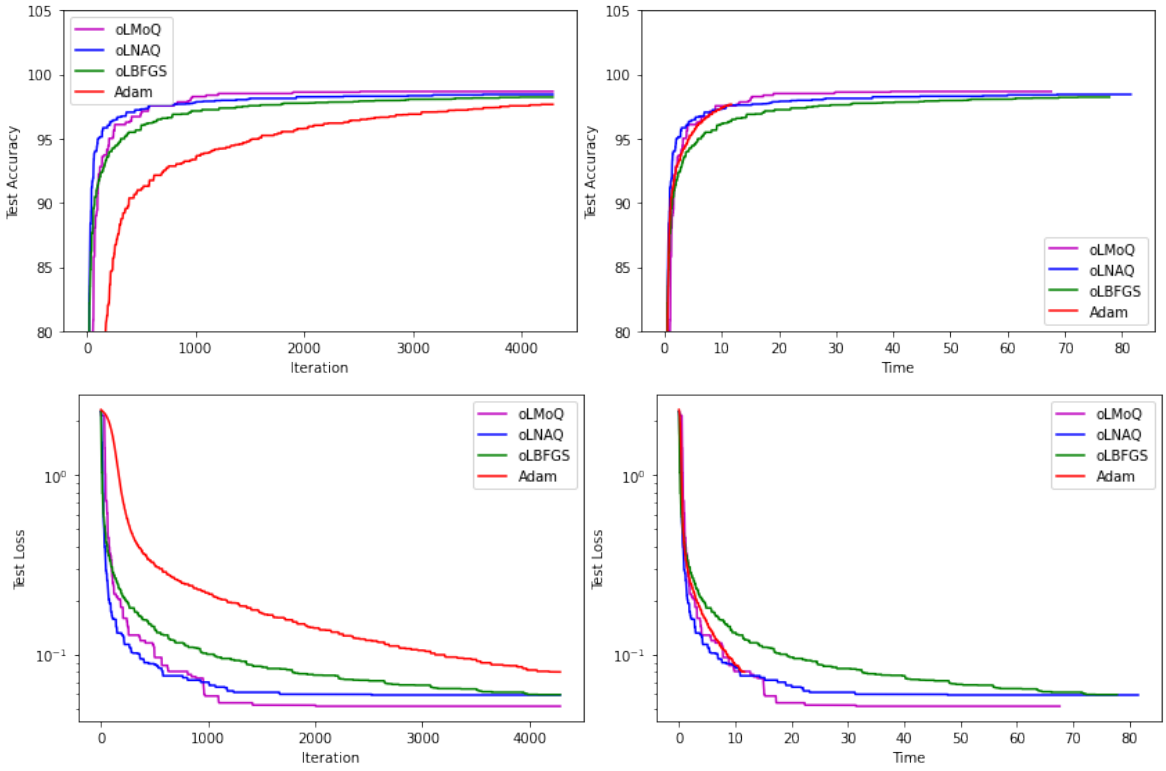


Figure 4.10: LeNET-5 results on 28×28 MNIST with $b = 128$.

4.5 Convergence Analysis

Suppose that $E : \mathbb{R}^d \to \mathbb{R}$ is continuosly differentiable and that $\mathbf{d} \in \mathbb{R}^d$, then from Taylor series, the quadratic model of the objective function at an iterate $\mathbf{w}_k$ is given as

$$E(\mathbf{w}_k + \mathbf{d}) \approx m_k(\mathbf{d}) \approx E(\mathbf{w}_k) + \nabla E(\mathbf{w}_k)^{\mathrm{T}}\mathbf{d} + \frac{1}{2}\,\mathbf{d}^{\mathrm{T}}\nabla^2 E(\mathbf{w}_k)\mathbf{d}. \tag{4.11}$$

In order to find the minimizer $\mathbf{d}_k$, we equate $\nabla \mathbf{m}_k(\mathbf{d}) = 0$ and thus have

$$\mathbf{d}_k = -\nabla^2 E(\mathbf{w}_k)^{-1}\nabla E(\mathbf{w}_k) = -\mathbf{B}_k^{-1}\nabla E(\mathbf{w}_k)\,. \tag{4.12}$$

The new iterate $\mathbf{w}_{k+1}$ is given as,

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{B}_k^{-1}\nabla E(\mathbf{w}_k), \tag{4.13}$$

and the quadratic model at the new iterate is given as

$$E(\mathbf{w}_{k+1} + \mathbf{d}) \approx m_{k+1}(\mathbf{d}) \approx E(\mathbf{w}_{k+1}) + \nabla E(\mathbf{w}_{k+1})^{\mathrm{T}}\mathbf{d} + \frac{1}{2}\,\mathbf{d}^{\mathrm{T}}\mathbf{B}_{k+1}\mathbf{d}\,, \tag{4.14}$$

where α_k is the step length and $\mathbf{B}_k^{-1} = \mathbf{H}_k$ and its consecutive updates $\mathbf{B}_{k+1}^{-1} = \mathbf{H}_{k+1}$ are symmetric positive definite matrices satisfying the secant condition. The Nesterov's acceleration approximates the quadratic model at $\mathbf{w}_k + \mu_k \mathbf{v}_k$ instead of the iterate at $\mathbf{w}_k$. Here $\mathbf{v}_k = \mathbf{w}_k - \mathbf{w}_{k-1}$ and μ_k is the momentum coefficient in the range $(0, 1)$. Thus we have the new iterate $\mathbf{w}_{k+1}$ given as,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu_k \mathbf{v}_k - \alpha_k \mathbf{B}_k^{-1} \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k), \tag{4.15}$$

$$= \mathbf{w}_k + \mu_k \mathbf{v}_k + \alpha_k \mathbf{d}_k. \tag{4.16}$$

In order to show that the Nesterov accelerated updates also satisfy the secant condition, we require that the gradient of m_{k+1} should match the gradient of the objective function at the last two iterates $(\mathbf{w}_k + \mu_k \mathbf{v}_k)$ and $\mathbf{w}_{k+1}$. In other words, we impose the following two requirements on $\mathbf{B}_{k+1}$,

$$\nabla m_{k+1}|_{\mathbf{d}=0} = \nabla E(\mathbf{w}_{k+1} + \mathbf{d})|_{\mathbf{d}=0} = \nabla E(\mathbf{w}_{k+1}), \tag{4.17}$$

$$\begin{aligned} \nabla m_{k+1}|_{\mathbf{d}=-\alpha_k \mathbf{d}_k} &= \nabla E(\mathbf{w}_{k+1} + \mathbf{d})|_{\mathbf{d}=-\alpha_k \mathbf{d}_k} \\ &= \nabla E(\mathbf{w}_{k+1} - \alpha_k \mathbf{d}_k) \\ &= \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k). \end{aligned} \tag{4.18}$$

From (4.14),

$$\nabla m_{k+1}(\mathbf{d}) = \nabla E(\mathbf{w}_{k+1}) + \mathbf{B}_{k+1} \mathbf{d}. \tag{4.19}$$

Substituting $\mathbf{d} = 0$ in (4.19), the condition in (4.17) is satisfied. From (4.18) and substituting $\mathbf{d} = -\alpha_k \mathbf{d}_k$ in (4.19), we have

$$\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) = \nabla E(\mathbf{w}_{k+1}) - \alpha_k \mathbf{B}_{k+1} \mathbf{d}_k. \tag{4.20}$$

Substituting for $\alpha_k \mathbf{d}_k$ from (4.16) in (4.20), we get

$$\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) = \nabla E(\mathbf{w}_{k+1}) - \mathbf{B}_{k+1}(\mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k)). \tag{4.21}$$

Also from the MoQ approximation given in 4.7, we have

$$(1 + \mu_k)\nabla \mathbf{E}(\mathbf{w}_k) - \mu_k \nabla \mathbf{E}(\mathbf{w}_{k-1}) = \nabla E(\mathbf{w}_{k+1}) - \mathbf{B}_{k+1}(\mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k)). \tag{4.22}$$

On rearranging the terms, we have the secant condition

$$\mathbf{y}_k = \mathbf{B}_{k+1} \mathbf{s}_k, \tag{4.23}$$

where, $\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k) = \alpha_k \mathbf{d}_k$ (4.24)

$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$ (in case of NAQ) (4.25)

$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - (1 + \mu)\nabla E(\mathbf{w}_k) + \mu_k \nabla E(\mathbf{w}_{k-1})$ (in case of MoQ) (4.26)

For the ease of convergence analysis, we consider an alternative way of expressing the iterate update equations. From (4.15) we have,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu_k \mathbf{v}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k), \tag{4.27}$$

and

$$\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k). \tag{4.28}$$

Let $\mathbf{a}_k = \frac{\mathbf{v}_k}{\alpha_k}$. From (4.28) and (4.27), we have

$$\mathbf{a}_{k+1} = \mu_k \mathbf{a}_k - \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \alpha_k \mathbf{a}_k). \tag{4.29}$$

and

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mu_k \mathbf{a}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \alpha_k \mathbf{a}_k) \tag{4.30}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{a}_{k+1}. \tag{4.31}$$

Let $\hat{\mathbf{w}}_k = \mathbf{w}_k + \mu_k \alpha_k \mathbf{a}_k$. Hence,

$$\mathbf{a}_{k+1} = \mu_k \mathbf{a}_k - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_{k+1}). \tag{4.32}$$

and,

$$\hat{\mathbf{w}}_{k+1} = \mathbf{w}_{k+1} + \mu_k \alpha_k \mathbf{a}_{k+1} \tag{4.33}$$

$$= \mathbf{w}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k \mathbf{a}_{k+1} \tag{4.34}$$

$$= \hat{\mathbf{w}}_k - \mu_k \alpha_k \mathbf{a}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k \mathbf{a}_{k+1} \tag{4.35}$$

$$= \hat{\mathbf{w}}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k (\mathbf{a}_{k+1} - \mathbf{a}_k) \tag{4.36}$$

$$\hat{\mathbf{w}}_{k+1} = \hat{\mathbf{w}}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k \left[(\mu - 1)\mathbf{a}_k - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k)\right] \tag{4.37}$$

By recursively substituting for $\mathbf{a}_{k+1}$ and $\mathbf{a}_k$until $\mathbf{a}_0 = 0$ we have

$$\hat{\mathbf{w}}_{k+1} = \hat{\mathbf{w}}_k - \alpha_k \sum_{i=0}^{k} \left(\mu_k^{k-i} \mathbf{H}_i \nabla E(\hat{\mathbf{w}}_i)\right) + \mu_k \alpha_k \left[(1 - \mu_k) \sum_{i=0}^{k-1} \left(\mu_k^{k-i} \mathbf{H}_i \nabla E(\hat{\mathbf{w}}_i)\right) - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k)\right] \tag{4.38}$$

$$\hat{\mathbf{w}}_{k+1} \approx \hat{\mathbf{w}}_k - \alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) + \mu_k \alpha_k \left[(1 - \mu_k)\mathbf{H}_{k-1} \nabla E(\hat{\mathbf{w}}_{k-1}) - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k)\right] \tag{4.39}$$

$$\hat{\mathbf{w}}_{k+1} \approx \hat{\mathbf{w}}_k - \alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) - \mu_k \alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) \tag{4.40}$$

$$\hat{\mathbf{w}}_{k+1} \approx \hat{\mathbf{w}}_k - (1 + \mu_k)\alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) \tag{4.41}$$

Assumption 4.5.1. *The sequence of iterates $\mathbf{w}_k$ and $\hat{\mathbf{w}}_k$ $\forall\ k = 1, 2, ..., k_{max} \in \mathbb{N}$ remains in the closed and bounded set $\mathbf{\Omega}$ on which the stochastic objective function is twice continuously differentiable and has Lipschitz continuous stochastic gradient, i.e., there exists a constant $L > 0$ such that*

$$\|\nabla E_b(\hat{\mathbf{w}}_{k+1}) - \nabla E_b(\hat{\mathbf{w}}_k)\| \leq L\|\hat{\mathbf{w}}_{k+1} - \hat{\mathbf{w}}_k\| \qquad \forall\ \hat{\mathbf{w}}_k \in \mathbb{R}^d, \tag{4.42}$$

and the mini batch samples X_k are drawn independently and the stochastic (minibatch) gradient $\nabla E_b(\hat{\mathbf{w}}_k) = \nabla E(\hat{\mathbf{w}}_k, X_k)$ is an unbiased estimator of the full gradient for all $\hat{\mathbf{w}}_k$, i.e., $\mathbb{E}[\nabla E(\hat{\mathbf{w}}_k, X_k)] \approx \nabla E(\hat{\mathbf{w}}_k)$

Assumption 4.5.2. *The Hessian matrix $\mathbf{B}_k = \nabla^2 \mathbf{E}_b(\hat{\mathbf{w}}_k)$ constructed with mini-batch samples X_k is bounded and well-defined, .i.e, there exists constants ρ and L, such that*

$$\rho \preccurlyeq \|\mathbf{B}_k\| \preccurlyeq L \quad \forall\ \ k = 1, 2, ..., k_{max} \in \mathbb{N} \tag{4.43}$$

for all mini-batch samples $X_k \subset T_r$ of batch size b.

Assumption 4.5.3. *There exists a constant γ^2 such that $\forall\ \hat{\mathbf{w}}_k \in \mathbb{R}^d$ and batches $X_k \subset T_r$ of size b,*

$$\mathbb{E}_{X_k}\left[\|\nabla E(\hat{\mathbf{w}}_k, X_k)\|^2\right] \leq \gamma^2 \tag{4.44}$$

Assumption 4.5.4. *The sequence of step size α_k selected is nonsummnable but square summable i.e.,*

$$\sum_{k=0}^{\infty} \alpha_k = \infty \qquad \text{and} \qquad \sum_{k=0}^{\infty} \alpha_k^2 \leq \infty \tag{4.45}$$

Lemma 4.5.1. *Suppose Assumptions 4.5.1 - 4.5.2 hold, there exists constants $0 \leq \lambda_1 \leq \lambda_2$ such that the set of $\{\mathbf{H}_k\}$ generated by the algorithm in the stochastic form satisfies*

$$\delta_1 \preccurlyeq \mathbf{H}_k \preccurlyeq \delta_2 \tag{4.46}$$

Proof. For ease of analysis, we study with respect to the direct Hessian $\mathbf{B}_k$ instead of the inverse Hessian $\mathbf{H}_k$. The updates of proposed oLNAQ and oLMoQ given as

1. Set $\mathbf{B}_k^{(0)} = \frac{\mathbf{y}_k^{\mathrm{T}}\mathbf{y}_k}{\mathbf{s}_j^{\mathrm{T}}\mathbf{y}_k}\mathbf{I}$ and $m = \min\{k, m_L\}$, where m_L is the limited memory size.
2. For $i = 0, ..., m-1$, set $j = k - m + 1 + i$ and compute

$$\mathbf{B}_k^{(i+1)} = \mathbf{B}_k^{(i)} - \frac{\mathbf{B}_k^{(i)}\mathbf{s}_j\mathbf{s}_j^{\mathrm{T}}\mathbf{B}_k^{(i)}}{\mathbf{s}_j^{\mathrm{T}}\mathbf{B}_k^{(i)}\mathbf{s}_j} + \frac{\mathbf{y}_j\mathbf{y}_j^{\mathrm{T}}}{\mathbf{y}_j^{\mathrm{T}}\mathbf{s}_j} \tag{4.47}$$

3. Set $\mathbf{B}_{k+1} = \mathbf{B}_k^{(m_L)}$
4. Update the curvature pairs $\mathbf{s}_k$ and $\mathbf{y}_k$ as

$$\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu\mathbf{v}_k) = \mathbf{w}_{k+1} - \hat{\mathbf{w}}_k, \tag{4.48}$$

$$\mathbf{y}_k = \nabla E_b(\mathbf{w}_{k+1}) - \nabla E_b(\mathbf{w}_k + \mu\mathbf{v}_k) = \nabla E_b(\mathbf{w}_{k+1}) - \nabla E_b(\hat{\mathbf{w}}_k) \qquad \text{(in case of NAQ)} \tag{4.49}$$

$$\mathbf{y}_k = \nabla E_b(\mathbf{w}_{k+1}) - (1+\mu)\nabla E_b(\mathbf{w}_k) + \mu\nabla E_b(\mathbf{w}_{k-1}) \qquad \text{(in case of MoQ)} \tag{4.50}$$

As consequence of Assumption 4.5.2 and the convexity of the objective function defined in Assumption 4.5.1, the eigenvalues of any sub-sampled Hessian are bounded above and away from zero and we have,

$$\rho\mathbf{s}_k \leq \mathbf{B}_k\mathbf{s}_k \leq L\mathbf{s}_k \tag{4.51}$$

$$\rho\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \leq \mathbf{y}_k^{\mathrm{T}}\mathbf{B}_k\mathbf{s}_k \leq L\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \tag{4.52}$$

$$\rho\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \leq \mathbf{y}_k^{\mathrm{T}}\mathbf{y}_k \leq L\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \tag{4.53}$$

$$\rho\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \leq \|\mathbf{y}_k\|^2 \leq L\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \tag{4.54}$$

$$\rho \leq \frac{\|\mathbf{y}_k\|^2}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k} \leq L \tag{4.55}$$

Therefore $\mathbf{B}_k^{(0)}$ initialized by $\frac{\mathbf{y}_k^{\mathrm{T}}\mathbf{y}_k}{\mathbf{s}_j^{\mathrm{T}}\mathbf{y}_k}\mathbf{I}$ is bounded away from zero. The trace and determinant of a matrix gives the bound on the largest and smallest eigenvalues respectively. Hence we use the trace-determinant argument to show that the eigenvalues of $\mathbf{B}_k$ are bounded away from zero. Let $Tr\big(\mathbf{B}_k\big)$ and $Det\big(\mathbf{B}_k\big)$ be the trace and determinant of the Hessian $\mathbf{B}_k$.

$$Tr\big(\mathbf{B}_{k+1}\big) = Tr\big(\mathbf{B}_k^{(0)}\big) - Tr\sum_{i=1}^{m}\left(\frac{\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}}\right) + Tr\sum_{i=1}^{m}\left(\frac{\mathbf{y}_{k,i}\mathbf{y}_{k,i}^{\mathrm{T}}}{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}\right) \tag{4.56}$$

$$Tr\big(\mathbf{B}_{k+1}\big) \leq Tr\big(\mathbf{B}_k^{(0)}\big) + Tr\sum_{i=1}^{m}\left(\frac{\|\mathbf{y}_{k,i}\|^2}{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}\right) \tag{4.57}$$

$$Tr\big(\mathbf{B}_{k+1}\big) \leq Tr\big(\mathbf{B}_k^{(0)}\big) + mL \leq C_1, \tag{4.58}$$

for some positive constant C_1, where the inequalities are due to $\mathbf{B}_k^{(0)}$ being bounded away from zero (as seen above) and (4.55).

Using the result from Powell [61], the determinant of the Hessian $\mathbf{B}_{k+1}$ generated by the proposed stochastic NAQ and MoQ algorithms can be expressed as,

$$Det\big(\mathbf{B}_{k+1}\big) = Det\big(\mathbf{B}_k^{(0)}\big)\prod_{i=1}^{m}\frac{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}} \tag{4.59}$$

$$Det\big(\mathbf{B}_{k+1}\big) = Det\big(\mathbf{B}_k^{(0)}\big)\prod_{i=1}^{m}\frac{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}\frac{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}} \tag{4.60}$$

$$Det\big(\mathbf{B}_{k+1}\big) \geq Det\big(\mathbf{B}_k^{(0)}\big)\Big(\frac{\rho}{C_1}\Big)^m \geq C_2 \tag{4.61}$$

for some positive constant C_2, where the inequalities are due to the largest eigenvalues $\mathbf{B}_k^{(i)}$ is less than C_1 (as seen above) and Assumption 4.5.2.

From the above trace-determinant inequalities above, the eigenvalues of all $\mathbf{B}_k$ are bounded away from zero uniformly.

□

Lemma 4.5.2. *Suppose Assumptions 4.5.1 - 4.5.3 hold and* $\delta_1\mathbf{I} \leq \|\mathbf{H}_k\| \leq \delta_2\mathbf{I} \;\; \forall \; k = 1, 2, ..., k_{max} \in \mathbb{N}$ *and* $0 < \delta_1 \leq \delta_2$*, then the iterates* $\{\hat{\mathbf{w}}_k\}$ *and average function values* $E(\hat{\mathbf{w}}_k)$ *generated by the algorithm satisfies*

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1})] - E(\mathbf{w}^*) \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k\delta_1\|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \tfrac{L}{2}(1+\mu_k)^2\alpha_k^2\delta_2^2\gamma^2 \tag{4.62}$$

Proof.

$$E(\hat{\mathbf{w}}_{k+1}) = E(\hat{\mathbf{w}}_k - (1+\mu_k)\alpha_k\mathbf{H}_{\mathbf{k}}\nabla E_b(\hat{\mathbf{w}}_k)) \tag{4.63}$$

$$E(\hat{\mathbf{w}}_{k+1}) \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k\nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}}\mathbf{H}_{\mathbf{k}}\nabla E(\hat{\mathbf{w}}_k) + \tfrac{L}{2}\|\hat{\mathbf{w}}_{k+1} - \hat{\mathbf{w}}_k\|_2^2 \tag{4.64}$$

$$E(\hat{\mathbf{w}}_{k+1}) \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k\nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}}\mathbf{H}_{\mathbf{k}}\nabla E_b(\hat{\mathbf{w}}_k) + \tfrac{L}{2}\|(1+\mu_k)\alpha_k\mathbf{H}_{\mathbf{k}}\nabla E_b(\hat{\mathbf{w}}_k)\|_2^2 \tag{4.65}$$

$$E(\hat{\mathbf{w}}_{k+1}) \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k\nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}}\mathbf{H}_{\mathbf{k}}\nabla E_b(\hat{\mathbf{w}}_k) + \tfrac{L}{2}(1+\mu_k)^2\alpha_k^2\delta_2^2\|\nabla E_b(\hat{\mathbf{w}}_k)\|_2^2 \tag{4.66}$$

Taking the expectation over all X_k we have,

$$\mathbb{E}_{X_k}\left[E(\hat{\mathbf{w}}_{k+1})\right] \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}} \mathbf{H_k} \nabla E(\hat{\mathbf{w}}_k) + \frac{L}{2}(1+\mu_k)^2 \alpha_k^2 \delta_2^2 \mathbb{E}_{X_k}\left[\|\nabla E_b(\hat{\mathbf{w}}_k)\|_2\right]^2 \tag{4.67}$$

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1})] \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k \delta_1 \|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \frac{L}{2}(1+\mu_k)^2 \alpha_k^2 \delta_2^2 \gamma^2 \tag{4.68}$$

Subtracting the optimal objective function value $E(\mathbf{w}^*)$ from both sides,

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k \delta_1 \|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \frac{L}{2}(1+\mu_k)^2 \alpha_k^2 \delta_2^2 \gamma^2 \tag{4.69}$$

□

Theorem 4.5.1. *Suppose Assumptions 4.5.1 - 4.5.4 and Lemma 2 holds true, then,*

$$\lim_{k\to\infty} \|\nabla E(\hat{\mathbf{w}}_k)\| = 0. \tag{4.70}$$

Proof. The proof uses Lemma 2 to build a supermartingale sequence similar to [46] (see Theorem 6 proof). We define the stochastic process ζ_k as

$$\zeta_k := E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) + \frac{L}{2}(1+\mu_k)^2 \delta_2^2 \gamma^2 \sum_{u=k}^{\infty} \alpha_u^2 \tag{4.71}$$

From Assumption 4.5.4, ζ_k is well defined as $\sum_{u=k}^{\infty} \alpha_u^2 < \sum_{u=0}^{\infty} \alpha_u^2 < \infty$ is summable. Further we define the sequence β_k as

$$\beta_k := (1+\mu_k)\alpha_k \delta_1 \|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.72}$$

The conditional expectation of ζ_k given $\mathcal{F}_k$, where $\mathcal{F}_k$ is the σ-algebra measuring ζ_k, β_k and $\hat{\mathbf{w}}_k$, is given as

$$\mathbb{E}[\zeta_{k+1}|\mathcal{F}_k] = \mathbb{E}[E(\hat{\mathbf{w}}_{k+1})|\mathcal{F}_k] - E(\mathbf{w}^*) + \frac{L}{2}(1+\mu_k)^2 \delta_2^2 \gamma^2 \sum_{u=k+1}^{\infty} \alpha_u^2 \tag{4.73}$$

Substituting 4.69 in 4.73 and using the definitions of ζ_k and β_k,

$$\mathbb{E}[\zeta_{k+1}|\zeta_k] \leq \zeta_k - \beta_k \tag{4.74}$$

Since the sequence of ζ_k and β_k are non-negative, they satisfy the conditions of the supermartingale convergence theorem. Therefore we can conclude that sequence ζ_k converges almost surely and the sum $\sum_{k=0}^{\infty} \beta_k < \infty$ is almost surely finite. Therefore we have

$$\sum_{k=0}^{\infty} (1+\mu_k)\alpha_k \delta_1 \|\nabla E(\hat{\mathbf{w}}_k)\|^2 < \infty \tag{4.75}$$

Since, $(1+\mu_k)\delta_1$ is constant and the sequence of step sizes α_k is non-summable, it implies that

$$\lim_{k\to\infty} \|\nabla E(\hat{\mathbf{w}}_k)\| = 0. \tag{4.76}$$

□

Theorem 4.5.2. *Suppose Assumptions 4.5.1 - 4.5.4 and Lemma 2 holds true, let* $\mathbf{w}^*$ *be the unique minimizer of the objective function, then for all k we have,*

$$E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*) \leq \eta^k E(\hat{\mathbf{w}}_0) - E(\mathbf{w}^*) \tag{4.77}$$

where the convergence rate η is linear and is given by

$$\eta = \left(1 - L(1+\mu)\alpha_k\left[2\delta_1 + L(1+\mu)\alpha_k\delta_2^2\right]\right) < 1 \tag{4.78}$$

provided the step size α_k is chosen by a decaying schedule.

Proof.

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k\delta_1\|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \tfrac{L}{2}(1+\mu_k)^2\alpha_k^2\delta_2^2\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.79}$$

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k\left[\delta_1 + \tfrac{L}{2}(1+\mu_k)\alpha_k\delta_2^2\right]\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.80}$$

For any vector $\mathbf{p} \in \mathbb{R}^d$, from optimality condition we have,

$$E(\mathbf{p}) \geq E(\hat{\mathbf{w}}_k) + \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}}(\mathbf{p} - \hat{\mathbf{w}}_k) + \tfrac{L}{2}\|\mathbf{p} - \hat{\mathbf{w}}_k\|^2 \tag{4.81}$$

$$\geq E(\hat{\mathbf{w}}_k) + \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}}(-\tfrac{1}{\rho}\nabla E(\hat{\mathbf{w}}_k)) + \tfrac{L\rho}{2}\|\tfrac{1}{L}\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.82}$$

$$\geq E(\hat{\mathbf{w}}_k) - \tfrac{1}{L}\|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \tfrac{L}{2}\|\tfrac{1}{L}\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.83}$$

$$\geq E(\hat{\mathbf{w}}_k) - \tfrac{1}{2L}\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.84}$$

Setting $\mathbf{p} = \mathbf{w}^*$,

$$E(\mathbf{w}^*) \geq E(\hat{\mathbf{w}}_k) - \tfrac{1}{2L}\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.85}$$

$$2L[E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*)] \leq \|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{4.86}$$

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k\left[\delta_1 + \tfrac{L}{2}(1+\mu_k)\alpha_k\delta_2^2\right]2L[E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*)] \tag{4.87}$$

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq [E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*)]\left[1 - L(1+\mu_k)\alpha_k\left[2\delta_1 + L(1+\mu_k)\alpha_k\delta_2^2\right]\right] \tag{4.88}$$

We define $\phi_k = \mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*)]$

$$\phi_{k+1} \leq \phi_k\left(1 - L(1+\mu_k)\alpha_k\left[2\delta_1 + L(1+\mu_k)\alpha_k\delta_2^2\right]\right) \tag{4.89}$$

By recursive application of the above inequality, we get,

$$\phi_{k+1} \leq \phi_0\left(1 - L(1+\mu)\alpha_k\left[2\delta_1 + L(1+\mu)\alpha_k\delta_2^2\right]\right)^k \tag{4.90}$$

The convergence rate is given as

$$\frac{E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)}{E(\hat{\mathbf{w}}_0) - E(\mathbf{w}^*)} \leq \left(1 - L(1+\mu)\alpha_k\left[2\delta_1 + L(1+\mu)\alpha_k\delta_2^2\right]\right)^k \tag{4.91}$$

□

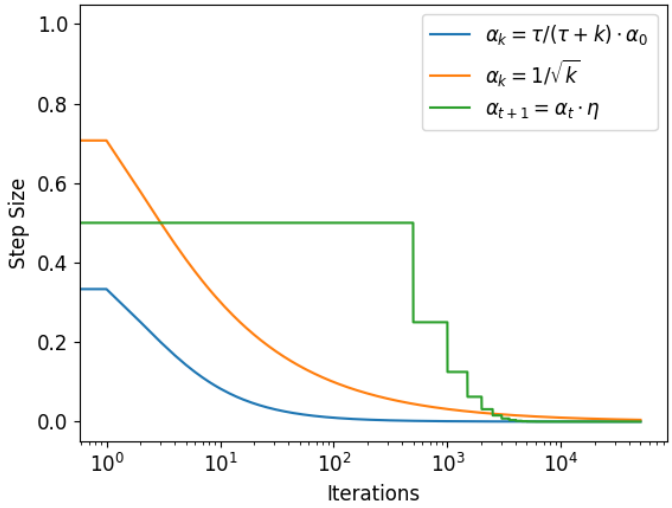


Figure 4.11: A comparison of the step size decay schedules.

4.6 Discussions

4.6.1 Choice of step size

In full batch methods, the step size or the learning rate is usually determined by line search methods satisfying either Armijo or Wolfe conditions. However, in stochastic methods, line searches are not quite effective since search conditions apply global validity. This cannot be assumed when using small local sub-samples [44]. Several studies show that line search methods does not necessarily ensure global convergence and have proposed methods that eliminate line search [27, 54, 55]. Moreover, determining step size using line search methods involves additional function computations until the search conditions such as the Armijo or Wolfe condition is satisfied. Hence we determine the step size using a simple learning rate schedule. Common learning rate schedules are polynomial decays and exponential decay functions. In this study, we determine the step size of o(L)NAQ using (4.92), o(LBFGS) using (4.93) and oLMoQ using (4.94)

$$\alpha_k = 1/\sqrt{k}, \tag{4.92}$$

$$\alpha_k = \tau/(\tau + k) \cdot \alpha_0, \tag{4.93}$$

$$\alpha_{t+1} = \alpha_t \cdot \eta, \tag{4.94}$$

where t is the epoch number, α_0 is set to 1 and η is a chosen between (0,1), recommended setting is 0.5. If the step size is too large, which is the case in the initial iterations, the learning can become unstable. This is stabilized by direction normalization and applied to all oLBFGS, oLNAQ and oLMoQ.

4.6.2 Choice of parameters

The momentum term μ is a hyperparameter with a value in the range $0 < \mu < 1$ and is usually chosen closer to 1 [19, 31]. The performance for different values of the momentum term were studied for both the classification and regression datasets. For the limited memory schemes, a memory size of $m = 4$

showed optimum results for all the four problem datasets with different batch sizes. Larger memory sizes also show good performance. However considering computational efficiency, memory size is usually maintained smaller than the batch size. Since the computation cost is $2bd + 6md$, if $b \approx m$ the computation cost would increase to $8bd$. Hence a smaller memory is desired. Memory sizes less than $m = 4$ does not perform well for small batch sizes and hence $m = 4$ was chosen.

4.6.3 Computation and Storage Cost

Table 4.1: Summary of Computational Cost and Storage.

	Algorithm	Computational Cost	Storage
full batch	BFGS	$nd + d^2 + \zeta nd$	d^2
	NAQ	$2nd + d^2 + \zeta nd$	d^2
	MoQ	$nd + d^2 + \zeta nd$	$d^2 + d$
	LBFGS	$nd + 4md + 2d + \zeta nd$	$2md$
	LNAQ	$2nd + 4md + 2d + \zeta nd$	$2md$
	LMoQ	$nd + 4md + 2d + \zeta nd$	$(2m + 1)d$
online	oBFGS	$2bd + d^2$	d^2
	oNAQ	$2bd + d^2$	d^2
	oMoQ	$bd + d^2$	$d^2 + d$
	oLBFGS	$2bd + 6md$	$2md$
	oLNAQ	$2bd + 6md$	$2md$
	oLMoQ	$bd + 6md$	$(2m + 1)d$

The summary of the computational cost and storage for full batch and stochastic (online) methods are illustrated in Table 4.1. The cost of function and gradient evaluations can be considered to be nd, where n is the number of training samples involved and d is the number of parameters. The Nesterov's Accelerated quasi-Newton (NAQ) method computes the gradient twice per iteration compared to the BFGS quasi-Newton method which computes the gradient only once per iteration. Thus NAQ has an additional nd computation cost. The MoQ method approximates the Nesterov's accelerated gradient as a linear combination of past gradients, thereby computing only one gradient per iteration like the BFGS method. In BFGS, NAQ and MoQ algorithms, the step length is determined by line search methods which involves ζ function evaluations until the search condition is satisfied. The $k - 1^{th}$ gradient is stored in memory. In the limited memory forms the Hessian update is approximated using the two-loop recursion scheme, which requires $4md + 2d$ multiplications. In the stochastic setting, both oBFGS and oNAQ compute the gradient twice per iteration, making the computational cost the same in both. On the other hand, oMoQ computes only one gradient per iteration, thus reducing the computation cost by nd compared to oBFGS and oNAQ. The stochastic methods do not use line search and due to smaller number of training samples (minibatch) in each iteration, the computational cost is smaller compared to full batch. Further, in stochastic limited memory methods, an additional $2md$ evaluations are required to compute the search direction as given (4.3). In stochastic methods the computational complexity is reduced significantly due to smaller batch sizes ($b < n$).

4.7 Summary

In this chapter we have proposed the stochastic extensions of the momentum and Nesterov's accelerated quasi-Newton method, in its full and limited memory forms, namely o(L)NAQ and o(LMoQ). The proposed algorithms is shown to be efficient compared to the stochastic o(L)BFGS method, thus confirming the effect of acceleration due to the momentum and Nesterov's gradient terms even in stochastic settings. We introduced direction normalization to ensure better stability in stochastic updates. From the results presented above, we can conclude that the proposed o(L)MoQ methods performs better than conventional o(L)BFGS methods and on par with o(L)NAQ with a reduced computation cost as a result of only one gradient computation per iteration. It must also be noted that the proposed o(L)MoQ method may be subject to stochastic noise as the curvature information is estimated from gradients computed on different mini-batch samples. In any case, our o(L)MoQ method performs better than the o(L)BFGS. Further analysis on the stochastic noise incurred is kept for future works. In the future, the effectiveness of the proposed o(L)MoQ will be studied on larger problems. Also, a detailed study on the effect of the limited memory size m, choice of momentum parameter and learning rate scheme can be studied in future works.

5

Adaptive Stochastic Nesterov's Accelerated quasi-Newton

A common problem in training neural networks is the vanishing and/or exploding gradient problem [62] which is more prominently seen in training of Recurrent Neural Networks (RNNs). This chapter proposes an adaptive stochastic Nesterov's accelerated quasi-Newton (aSNAQ) method for training RNNs. The proposed method is an accelerated second-order method that attempts to incorporate curvature information while maintaining a low per iteration cost. Furthermore, direction normalization has been introduced to solve the vanishing and/or exploding gradient problem. This chapter is based on the results published in [63] and [64]. In addition, we also discuss the convergence of the proposed aSNAQ method.

5.1 Introduction

Neural networks have shown to be effective in several applications. However, neural network training poses several challenges. A common problem in training neural networks is the vanishing and/or exploding gradient problem which is more prominently seen in the training of Recurrent Neural Networks. Recurrent Neural Networks (RNNs) are powerful sequence models, popularly used in solving pattern recognition and sequence modeling problems such as text generation, image captioning, machine translation, speech recognition, etc. The structure of RNNs are similar to feedforward neural networks except that they also allow self-loops and backward connections between its nodes [65]. These connections make RNNs capable of learning, retaining and expressing long sequential relations.

Despite the capabilities of RNNs in modeling sequences, RNNs are particularly very difficult to train mainly due to the vanishing and/or exploding gradient problem [62]. Several algorithms and architectures have been proposed to address the issues involved in training RNNs [66]. Architectures such as Long Short-Term Memory (LSTM) [67] and Gated Recurrent Units (GRU) have shown to be more resilient to the gradient issues compared to vanilla RNNs. Several other studies revolve around proposing algorithms that can be effectively used in training RNNs, some of which propose the use of second-order curvature information [68]. Though first-order methods are popular for their simplicity and low computational complexity, second-order methods have shown to speed up convergence despite its high computational cost. However, very few attempts have been made to train RNNs using second-order methods. [69] proposed a variant of the Broyden-Fletcher-Goldfarb-Shanon (BFGS) method that incorporates adaptive mechanisms to train RNNs. However, the high computational cost incurred

in second-order methods still poses a major challenge, which further adds up in very long sequence modeling problems. Recent studies [66, 68, 70, 71] propose algorithms that judiciously incorporate curvature information while taking the computation cost into consideration.

5.2 Background

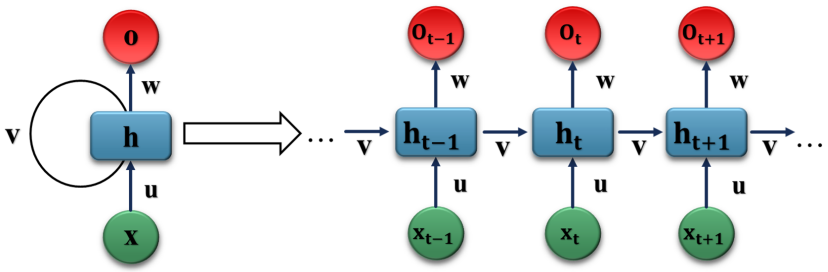


Figure 5.1: Structure of a recurrent neural network.

Given an input sequence $\mathbf{x} = \{x_1, ..., x_\tau\}$, RNNs use their internal states to process the sequence of inputs. The internal states $\mathbf{h}_t$ of the RNN and the output vector sequence $\hat{\mathbf{z}}_t$ can be formalized as

$$\mathbf{h}_t = \psi_h(\mathbf{x}_t \mathbf{w}_{xh} + \mathbf{h}_{t-1} \mathbf{w}_{hh} + \mathbf{b}_h), \tag{5.1}$$

$$\hat{\mathbf{z}}_t = \psi_o(\mathbf{h}_t \mathbf{w}_{ho} + \mathbf{b}_o), \tag{5.2}$$

where $\mathbf{w}_{xh}$ is the input to hidden weight matrix, $\mathbf{w}_{hh}$ is the hidden to hidden recurrent weight matrix, and $\mathbf{w}_{ho}$ is the hidden to output weight matrix. $\mathbf{b}_h$ and $\mathbf{b}_o$ are the bias vectors of the hidden and output nodes respectively. An activation function ψ such as tanh, ReLU, sigmoid or softmax is used to introduce non-linearity.

Training in neural networks is an iterative process in which the parameters (the weights and biases) are updated in order to minimize an objective function. Gradient based algorithms are popularly used for training and the gradients of the RNN are computed using backpropagation through time [72, 73]. Given a mini-batch $X \subseteq T_r$ with samples $(x_p, z_p)_{p \in X}$ drawn at random from the training set T_r and error function $E_p(\mathbf{w}; x_p, z_p)$ parameterized by a vector $\mathbf{w} = \{\mathbf{w}_{xh}, \mathbf{w}_{hh}, \mathbf{w}_{ho}, \mathbf{b}_h, \mathbf{b}_o\} \in \mathbb{R}^d$, the objective function is defined as

$$\min_{\mathbf{w} \in \mathbb{R}^d} E(\mathbf{w}) = \frac{1}{|T_r|} \sum_{p \in T_r} E_p(\mathbf{w}). \tag{5.3}$$

Here we define the error function $E_p(\mathbf{w})$ to be the cross entropy error given by,

$$E_p(\mathbf{w}) = -\sum_t \mathbf{z}_t \log \hat{\mathbf{z}}_t, \tag{5.4}$$

where $\mathbf{z}_t$ is the expected output and $\hat{\mathbf{z}}_t$ is the output predicted by the RNN at timestep t. In stochastic (mini-batch) methods, the objective function is minimized using $\nabla E_b(\mathbf{w})$, the gradient of the error function calculated on a mini-batch $X \subseteq T_r$ of batch size $b = |X|$ as shown below.

$$E_b(\mathbf{w}) = \frac{1}{b} \sum_{p \in X} E_p(\mathbf{w}). \tag{5.5}$$

In gradient based methods, the objective function $E(\mathbf{w})$ under consideration is minimized by updating the parameters $\mathbf{w}$ using the iterative formula

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_{k+1}, \tag{5.6}$$

where k is the iteration count and $\mathbf{v}_{k+1}$ is the update vector, which is defined for each algorithm. This chapter proposes an accelerated second-order method for training RNNs that build on the algorithmic framework of the adaQN methods. In the following section we describe the adaQN method as proposed by [71].

5.2.1 adaQN

adaQN is a recently proposed method which was shown to be suitable for training RNNs as well [71]. It builds on the algorithmic framework of SQN [47] by decoupling the iterate and update cycles. adaQN targets the vanishing/exploding gradient issue by initializing $\mathbf{H}_k^{(0)}$ in the two-loop recursion (Appendix A.1, step 7) based on the accumulated gradient information as shown below.

$$[H_k^{(0)}]_{ii} = \frac{1}{\sqrt{\sum_{j=0}^{k} \nabla E(\mathbf{w}_j)_i^2 + \epsilon}} \tag{5.7}$$

adaQN proposes the use of an accumulated Fisher Information matrix (aFIM) that stores the last m_F gradient vectors $\nabla E(\mathbf{w}_k)$. This is used in the computation of the $\mathbf{y}$ vector for Hessian approximation as

$$\mathbf{y} = \frac{1}{|F|} \sum_{i=1}^{|F|} F_i \cdot s \tag{5.8}$$

The curvature pairs are computed every L steps and stored in (S, Y) buffer only if they are sufficiently large. Further, adaQN performs a control condition by comparing the error at current and previous aggregated weights on a monitoring dataset. If the current error is larger than the previous error by a factor γ, the aFIM and curvature pair buffers are cleared and the weights are reverted to the previous aggregated weights. This heuristic further avoids deterioration of performance due to noisy or stale curvatures.

5.3 Proposed aSNAQ Method

The proposed method - adaptive stochastic Nesterov Accelerated Quasi-Newton (aSNAQ) is a stochastic QN method by combining (L)NAQ and adaQN which is built on the algorithmic framework of SQN. It incorporates Nesterov's accelerated gradient term and a simple adaptively tuned momentum term. The algorithm is shown in Algorithm 6.1. In the limited memory scheme, the initialization of the Hessian $\mathbf{H}_k^{(0)}$ is important as the estimate of the Hessian approxmiation $\mathbf{H}_k$ is built upon $\mathbf{H}_k^{(0)}$. The most common initialiation used in most limited BFGS methods is a simple scalar initialization Hessian $\mathbf{H}_k^{(0)} = \frac{\mathbf{s}_k^T \mathbf{y}_k}{\mathbf{y}_k^T \mathbf{y}_k} \mathbf{I}$, which is not effective in solving the problems of RNNs. aSNAQ, like adaQN initializes the Hessian $\mathbf{H}_k^{(0)}$ based on accumulated gradient information as shown below.

$$[H_k^{(0)}]_{ii} = \frac{1}{\sqrt{\sum_{j=0}^{k} \nabla E(\mathbf{w}_j)_i^2 + \epsilon}} \tag{5.9}$$

Algorithm 5.1 adaQN Method

Require: minibatch X_k, aFIM buffer F of size m_F and curvature pair buffer (S, Y) of size m_L
Initialize: $\mathbf{w}_o$=$\mathbf{w}_k \in \mathbb{R}^d$, $\mathbf{w}_s = 0$, $k = 0$ and $t = 0$

1: **while** $k < k_{max}$ **do**
2: Calculate $\nabla E(\mathbf{w}_k)$
3: Determine $\mathbf{g}_k$ using $\nabla E(\mathbf{w}_k)$ in two-loop recursion (Appendix A.1)
4: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \alpha_k \mathbf{g}_k$
5: Store $\nabla E(\mathbf{w}_k)$ in F
6: $\mathbf{w}_s = \mathbf{w}_s + \mathbf{w}_{k+1}$ ▷ Weight aggregation
7: **if** mod(k , L) = 0 **then**
8: Compute average $\mathbf{w}_n = \mathbf{w}_s / L$
9: $\mathbf{w}_s = 0$
10: **if** $t > 0$ **then**
11: **if** $E(\mathbf{w}_n) > \gamma E(\mathbf{w}_o)$ **then**
12: Clear (S, Y) and F buffers
13: Reset $\mathbf{w}_k = \mathbf{w}_o$
14: **continue**
15: **end if**
16: $\mathbf{s} = \mathbf{w}_n - \mathbf{w}_o$
17: $\mathbf{y} = \frac{1}{|F|}(\sum_{i=1}^{|F|} F_i \cdot \mathbf{s})$
18: **if** $\mathbf{s}^{\mathrm{T}}\mathbf{y} > \epsilon \cdot \mathbf{s}^{\mathrm{T}}\mathbf{s}$ **then**
19: Store curvature pairs (**s**,**y**) in (S, Y)
20: **end if**
21: **end if**
22: Update $\mathbf{w}_o = \mathbf{w}_n$
23: $t \leftarrow t + 1$
24: **end if**
25: $k \leftarrow k + 1$
26: **end while**

The search direction vector $\mathbf{g}_k$ is computed using the two-loop recursion, incorporating the Nesterov's gradient as

$$\mathbf{g}_k = -\mathbf{H}_k \nabla E(\mathbf{w}_k + \mu \mathbf{v}_k). \tag{5.10}$$

An accumulated Fisher Information Matrix (aFIM) is used to store $\nabla E(\mathbf{w}_k) \nabla E(\mathbf{w}_k)^{\mathrm{T}}$ matrix in a FIFO memory buffer F of size m_F at each iteration. The curvature information pair $\{\mathbf{s}, \mathbf{y}\}$ is constructed using aFIM as shown in (5.36) and (5.37)

$$\mathbf{s} = \mathbf{w}_t - (\mathbf{w}_{t-1} + \mu \mathbf{v}_{t-1}), \tag{5.11}$$

$$\mathbf{y} = \frac{1}{|\mathbf{F}|} \sum_{i=1}^{|\mathbf{F}|} \mathbf{F}_i \cdot \mathbf{s}\,, \tag{5.12}$$

where $\mathbf{w}_t$ is the average aggregated weight, t is the curvature pair update counter, $\mathbf{F}_i = \nabla \mathcal{L}(\mathbf{w}_{k+1}) \nabla \mathcal{L}(\mathbf{w}_{k+1})^{\mathrm{T}}$ and $|\mathbf{F}|$ is the number of entries present in $\mathbf{F}$. In aSNAQ, the gradient at $\mathbf{w}_k + \mu \mathbf{v}_k$ is used in the search direction while the gradient at $\mathbf{w}_{k+1}$ is stored in the aFIM for Hessian approximation. Thus twice gradient computation per iteration is involved just like in NAQ. The curvature pairs are computed every L steps and stored in (S, Y) only if sufficiently large. The momentum term μ is tuned by a momentum update factor ϕ as shown in step 22. aSNAQ also performs a error control check as shown in step

14-18. In addition to reseting the aFIM and curvature pair buffers and restoring old parameters, the momentum is also scaled down (step17). Unlike adaQN the error control check is carried out on the same mini-batch sub-sample. Further, direction normalization [49] is introduced in step 4 to improve stability and to solve the exploding gradient issue.

Algorithm 5.2 aSNAQ Method

Require: minibatch X_k, μ_{min}, μ_{max}, k_{max}, accumulated Fisher Information Matrix (aFIM) buffer F of size m_F and curvature pair buffer (S, Y) of size m_L , momentum update factor ϕ

Initialize: $\mathbf{w}_k \in \mathbb{R}^d$, $\mu = \mu_{min}$, $\mathbf{v}_k$, $\mathbf{w}_o$, $\mathbf{v}_o$, $\mathbf{w}_s$, $\mathbf{v}_s$, k and $t = 0$

1: **while** $k < k_{max}$ **do**
2: Calculate $\nabla E(\mathbf{w}_k + \mu \mathbf{v}_k)$
3: Determine $\mathbf{g}_k$ using Algorithm A.1
4: $\mathbf{g}_k = \mathbf{g}_k / \|\mathbf{g}_k\|_2$ ▷ Direction normalization
5: $\mathbf{v}_{k+1} \leftarrow \mu \mathbf{v}_k + \alpha_k \mathbf{g}_k$
6: $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
7: Calculate $\nabla E(\mathbf{w}_{k+1})$ and store in F
8: $\mathbf{w}_s = \mathbf{w}_s + \mathbf{w}_k$
9: $\mathbf{v}_s = \mathbf{v}_s + \mathbf{v}_k$
10: **if** mod(k , L) $= 0$ **then**
11: Compute average $\mathbf{w}_t = \mathbf{w}_s / L$ and $\mathbf{v}_t = \mathbf{v}_s / L$
12: $\mathbf{w}_s = 0$ and $\mathbf{v}_s = 0$
13: **if** $t > 0$ **then**
14: **if** $E(\mathbf{w}_t) > \gamma E(\mathbf{w}_{t-1})$ **then**
15: Clear (S, Y) and F buffers
16: Reset $\mathbf{w}_k = \mathbf{w}_{t-1}$ and $\mathbf{v}_k = \mathbf{v}_{t-1}$
17: Update $\mu = \max(\mu / \phi, \mu_{min})$
18: **continue**
19: **end if**
20: $\mathbf{s} = \mathbf{w}_t - (\mathbf{w}_{t-1} + \mu \mathbf{v}_{t-1})$
21: $\mathbf{y} = \frac{1}{|F|}(\sum_{i=1}^{|F|} \mathbf{F_i} \cdot \mathbf{s})$
22: Update $\mu = \min(\mu \cdot \phi, \mu_{max})$
23: **if** $\mathbf{s}^T\mathbf{y} > \epsilon \cdot \mathbf{s}^T\mathbf{s}$ **then**
24: Store curvature pairs (**s**,**y**) in (S, Y)
25: **end if**
26: **end if**
27: $t \leftarrow t + 1$
28: **end if**
29: $k \leftarrow k + 1$
30: **end while**

5.4 Convergence Analysis

In this section, we give the convergence analysis of the proposed aSNAQ algorithm. As shown in the previous chapter, the Nesterov's accelerated updates satisfy the secant condition

$$\mathbf{y}_k = \mathbf{B}_{k+1}\mathbf{s}_k, \tag{5.13}$$

where,

$$\mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k) = \alpha_k \mathbf{d}_k \tag{5.14}$$

$$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) \tag{5.15}$$

For the ease of convergence analysis, we begin with the alternative expression of the iterate update equations as introduced in Chapter 4. From (4.15) we have,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu_k \mathbf{v}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k), \tag{5.16}$$

and

$$\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k). \tag{5.17}$$

Let $\mathbf{a}_k = \frac{\mathbf{v}_k}{\alpha_k}$. From (5.17) and (5.16), we have

$$\mathbf{a}_{k+1} = \mu_k \mathbf{a}_k - \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \alpha_k \mathbf{a}_k). \tag{5.18}$$

and

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mu_k \mathbf{a}_k - \alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k + \mu_k \alpha_k \mathbf{a}_k) \tag{5.19}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{a}_{k+1}. \tag{5.20}$$

Let $\hat{\mathbf{w}}_k = \mathbf{w}_k + \mu_k \alpha_k \mathbf{a}_k$. Hence,

$$\mathbf{a}_{k+1} = \mu_k \mathbf{a}_k - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_{k+1}). \tag{5.21}$$

and,

$$\hat{\mathbf{w}}_{k+1} = \mathbf{w}_{k+1} + \mu_k \alpha_k \mathbf{a}_{k+1} \tag{5.22}$$

$$= \mathbf{w}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k \mathbf{a}_{k+1} \tag{5.23}$$

$$= \hat{\mathbf{w}}_k - \mu_k \alpha_k \mathbf{a}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k \mathbf{a}_{k+1} \tag{5.24}$$

$$= \hat{\mathbf{w}}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k (\mathbf{a}_{k+1} - \mathbf{a}_k) \tag{5.25}$$

$$\hat{\mathbf{w}}_{k+1} = \hat{\mathbf{w}}_k + \alpha_k \mathbf{a}_{k+1} + \mu_k \alpha_k \left[(\mu - 1)\mathbf{a}_k - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k)\right] \tag{5.26}$$

$$\hat{\mathbf{w}}_{k+1} = \hat{\mathbf{w}}_k - \alpha_k \sum_{i=0}^{k} \left(\mu_k^{k-i} \mathbf{H}_i \nabla E(\hat{\mathbf{w}}_i)\right) + \mu_k \alpha_k \left[(1 - \mu_k) \sum_{i=0}^{k-1} \left(\mu_k^{k-i} \mathbf{H}_i \nabla E(\hat{\mathbf{w}}_i)\right) - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k)\right] \tag{5.27}$$

$$\hat{\mathbf{w}}_{k+1} \approx \hat{\mathbf{w}}_k - \alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) + \mu_k \alpha_k \left[(1 - \mu_k)\mathbf{H}_{k-1} \nabla E(\hat{\mathbf{w}}_{k-1}) - \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k)\right] \tag{5.28}$$

$$\hat{\mathbf{w}}_{k+1} \approx \hat{\mathbf{w}}_k - \alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) - \mu_k \alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) \tag{5.29}$$

$$\hat{\mathbf{w}}_{k+1} \approx \hat{\mathbf{w}}_k - (1 + \mu_k)\alpha_k \mathbf{H}_k \nabla E(\hat{\mathbf{w}}_k) \tag{5.30}$$

Assumption 5.4.1. *The sequence of iterates $\mathbf{w}_k$ and $\hat{\mathbf{w}}_k$ $\forall\ k = 1, 2, ..., k_{max} \in \mathbb{N}$ remains in the closed and bounded set $\boldsymbol{\Omega}$ on which the stochastic objective function is twice continuously differentiable and has Lipschitz continuous stochastic gradient, i.e., there exists a constant $L > 0$ such that*

$$\|\nabla E_b(\hat{\mathbf{w}}_{k+1}) - \nabla E_b(\hat{\mathbf{w}}_k)\| \leq L\|\hat{\mathbf{w}}_{k+1} - \hat{\mathbf{w}}_k\| \qquad \forall\ \hat{\mathbf{w}}_k \in \mathbb{R}^d, \tag{5.31}$$

and the mini batch samples X_k are drawn independently and the stochastic (minibatch) gradient $\nabla E_b(\hat{\mathbf{w}}_k) = \nabla E(\hat{\mathbf{w}}_k, X_k)$ is an unbiased estimator of the full gradient for all $\hat{\mathbf{w}}_k$, i.e., $\mathbb{E}[\nabla E(\hat{\mathbf{w}}_k, X_k)] \approx \nabla E(\hat{\mathbf{w}}_k)$

Assumption 5.4.2. *The Hessian matrix* $\mathbf{B}_k = \nabla^2 \mathbf{E}_b(\hat{\mathbf{w}}_k)$ *constructed with mini-batch samples* X_k *is bounded and well-defined, .i.e, there exists constants* ρ *and* L*, such that*

$$\rho \preccurlyeq \|\mathbf{B}_k\| \preccurlyeq L \quad \forall \ k = 1, 2, ..., k_{max} \in \mathbb{N} \tag{5.32}$$

for all mini-batch samples $X_k \subset T_r$ *of batch size b.*

Assumption 5.4.3. *There exists a constant* γ^2 *such that* $\forall\ \hat{\mathbf{w}}_k \in \mathbb{R}^d$ *and batches* $X_k \subset T_r$ *of size b,*

$$\mathbb{E}_{X_k}\left[\|\nabla E(\hat{\mathbf{w}}_k, X_k)\|^2\right] \leq \gamma^2 \tag{5.33}$$

Lemma 5.4.1. *Suppose Assumptions 5.4.1 - 5.4.2 hold, there exists constants* $0 \leq \lambda_1 \leq \lambda_2$ *such that the set of* $\{\mathbf{H}_k\}$ *generated by the algorithm in the stochastic form satisfies*

$$\delta_1 \preccurlyeq \mathbf{H}_k \preccurlyeq \delta_2 \tag{5.34}$$

Proof. For ease of analysis, we study with respect to the direct Hessian $\mathbf{B}_k$ instead of the inverse Hessian $\mathbf{H}_k$. The updates of proposed oLNAQ and oLMoQ given as

1. Set $\mathbf{B}_k^{(0)} = \sqrt{\sum_{j=0}^{k} \nabla E(\mathbf{w}_j)_i^2 + \epsilon}$ and $m = \min\{k, m_L\}$, where m_L is the limited memory size.
2. For $i = 0, ..., m-1$, set $j = k - m + 1 + i$ and compute

$$\mathbf{B}_k^{(i+1)} = \mathbf{B}_k^{(i)} - \frac{\mathbf{B}_k^{(i)} \mathbf{s}_j \mathbf{s}_j^{\mathrm{T}} \mathbf{B}_k^{(i)}}{\mathbf{s}_j^{\mathrm{T}} \mathbf{B}_k^{(i)} \mathbf{s}_j} + \frac{\mathbf{y}_j \mathbf{y}_j^{\mathrm{T}}}{\mathbf{y}_j^{\mathrm{T}} \mathbf{s}_j} \tag{5.35}$$

3. Set $\mathbf{B}_{k+1} = \mathbf{B}_k^{(m_L)}$
4. Update the curvature pairs $\mathbf{s}_k$ and $\mathbf{y}_k$ as

$$\mathbf{s}_k = \mathbf{w}_t - (\mathbf{w}_{t-1} + \mu \mathbf{v}_{t-1}) = \mathbf{w}_t - \hat{\mathbf{w}}_{t-1}, \tag{5.36}$$

$$\mathbf{y}_k = \frac{1}{|\mathbf{F}|} \sum_{i=1}^{|\mathbf{F}|} \mathbf{F}_i \cdot \mathbf{s}_k \quad . \tag{5.37}$$

The inverse Hessian is updated using the limited memory scheme, where the diagonal elements of the initial $H_k^{(0)}$ is given by

$$[H_k^{(0)}]_{ii} = \frac{1}{\sqrt{\sum_{j=0}^{k} \nabla E(\mathbf{w}_j)_i^2 + \epsilon}}, \tag{5.38}$$

$$[B_k^{(0)}]_{ii} = \sqrt{\sum_{j=0}^{k} \nabla E(\mathbf{w}_j)_i^2 + \epsilon}, \tag{5.39}$$

thus the eigenvalues of the matrix $B_k^{(0)}$ initialized by the diagonal matrix are bounded above zero, for all k.

The trace of a matrix gives the largest eigenvalue and determinant the smallest eigenvalue. Hence we use the trace-determinant argument to show that the eigenvalues of $\mathbf{B}_k$ are bounded away from zero. Let $Tr(\mathbf{B}_k)$ and $Det(\mathbf{B}_k)$ be the trace and determinant of the Hessian $\mathbf{B}_k$.

$$Tr\big(\mathbf{B}_{k+1}\big) = Tr\big(\mathbf{B}_k^{(0)}\big) - Tr\sum_{i=1}^{m}\left(\frac{\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}}\right) + Tr\sum_{i=1}^{m}\left(\frac{\mathbf{y}_{k,i}\mathbf{y}_{k,i}^{\mathrm{T}}}{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}\right) \tag{5.40}$$

$$Tr\big(\mathbf{B}_{k+1}\big) \leq Tr\big(\mathbf{B}_k^{(0)}\big) + Tr\sum_{i=1}^{m}\left(\frac{\|\mathbf{y}_{k,i}\|^2}{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}\right) \tag{5.41}$$

As consequence of Assumption 5.4.2 and the convexity of the objective function defined in Assumption 5.4.1, the eigenvalues of any sub-sampled Hessian are bounded above and away from zero and we have,

$$\rho\mathbf{s}_k \leq \mathbf{B}_k\mathbf{s}_k \leq L\mathbf{s}_k \tag{5.42}$$

$$\rho\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \leq \mathbf{y}_k^{\mathrm{T}}\mathbf{B}_k\mathbf{s}_k \leq L\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \tag{5.43}$$

$$\rho\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \leq \mathbf{y}_k^{\mathrm{T}}\mathbf{y}_k \leq L\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \tag{5.44}$$

$$\rho\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \leq \|\mathbf{y}_k\|^2 \leq L\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k \tag{5.45}$$

$$\rho \leq \frac{\|\mathbf{y}_k\|^2}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k} \leq L \tag{5.46}$$

Therefore, we have the trace of $\mathbf{B}_{k+1}$ bounded by some positive constant C_1 as

$$Tr\big(\mathbf{B}_{k+1}\big) \leq Tr\big(\mathbf{B}_k^{(0)}\big) + mL \leq C_1, \tag{5.47}$$

Using the result from Powell [61], the determinant of the Hessian $\mathbf{B}_{k+1}$ generated by the proposed stochastic NAQ and MoQ algorithms can be expressed as,

$$Det\big(\mathbf{B}_{k+1}\big) = Det\big(\mathbf{B}_k^{(0)}\big)\prod_{i=1}^{m}\frac{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}} \tag{5.48}$$

$$Det\big(\mathbf{B}_{k+1}\big) = Det\big(\mathbf{B}_k^{(0)}\big)\prod_{i=1}^{m}\frac{\mathbf{y}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}\frac{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{s}_{k,i}}{\mathbf{s}_{k,i}^{\mathrm{T}}\mathbf{B}_k^{(i)}\mathbf{s}_{k,i}} \tag{5.49}$$

$$Det\big(\mathbf{B}_{k+1}\big) \geq Det\big(\mathbf{B}_k^{(0)}\big)\Big(\frac{\rho}{C_1}\Big)^m \geq C_2 \tag{5.50}$$

for some positive constant C_2, where the inequalities are due to the largest eigenvalues $\mathbf{B}_k^{(i)}$ is less than C_1 (as seen above) and Assumption 5.4.2.

From the above trace-determinant inequalities above, the eigenvalues of all $\mathbf{B}_k$ are bounded away from zero uniformly.

□

Theorem 5.4.1. *Suppose Assumptions 5.4.1 - 5.4.3 hold and* $\delta_1\mathbf{I} \leq \|\mathbf{H}_k\| \leq \delta_2\mathbf{I} \;\; \forall\; k = 1, 2, ..., k_{max} \in \mathbb{N}$ *and* $0 < \delta_1 \leq \delta_2$*, then the iterates* $\{\hat{\mathbf{w}}_k\}$ *and average function values* $E(\hat{\mathbf{w}}_k)$ *generated by the algorithm for a constant step size* α_k *chosen such that* $0 \leq \alpha_k = (1+\mu_k)\alpha \leq \frac{\delta_1}{\delta_2^2 L}$ *converges to a stationary point* $\mathbf{w}^*$ *at a linear rate.*

$$E(\hat{\mathbf{w}}_{k+1})] - E(\mathbf{w}^*) \leq [1 - \alpha(1+\mu_k)\delta_1\rho]^k E(\hat{\mathbf{w}}_0) - E(\mathbf{w}^*) \tag{5.51}$$

Proof.

$$E(\hat{\mathbf{w}}_{k+1}) = E(\hat{\mathbf{w}}_k - (1+\mu_k)\alpha_k \mathbf{H_k} \nabla E_b(\hat{\mathbf{w}}_k)) \tag{5.52}$$

$$E(\hat{\mathbf{w}}_{k+1}) \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}} \mathbf{H_k} \nabla E(\hat{\mathbf{w}}_k) + \tfrac{L}{2}\|\hat{\mathbf{w}}_{k+1} - \hat{\mathbf{w}}_k\|_2^2 \tag{5.53}$$

$$E(\hat{\mathbf{w}}_{k+1}) \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}} \mathbf{H_k} \nabla E_b(\hat{\mathbf{w}}_k) + \tfrac{L}{2}\|(1+\mu_k)\alpha_k \mathbf{H_k} \nabla E_b(\hat{\mathbf{w}}_k)\|_2^2 \tag{5.54}$$

$$E(\hat{\mathbf{w}}_{k+1}) \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}} \mathbf{H_k} \nabla E_b(\hat{\mathbf{w}}_k) + \tfrac{L}{2}(1+\mu_k)^2\alpha_k^2\delta_2^2\|\nabla E_b(\hat{\mathbf{w}}_k)\|_2^2 \tag{5.55}$$

Taking the expectation over all X_k we have,

$$\mathbb{E}_{X_k}\left[E(\hat{\mathbf{w}}_{k+1})\right] \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}} \mathbf{H_k} \nabla E(\hat{\mathbf{w}}_k) + \tfrac{L}{2}(1+\mu_k)^2\alpha_k^2\delta_2^2 \mathbb{E}_{X_k}\left[\|\nabla E_b(\hat{\mathbf{w}}_k)\|_2\right]^2 \tag{5.56}$$

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1})] \leq E(\hat{\mathbf{w}}_k) - (1+\mu_k)\alpha_k\delta_1\|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \tfrac{L}{2}(1+\mu_k)^2\alpha_k^2\delta_2^2\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.57}$$

Subtracting the optimal objective function value $E(\mathbf{w}^*)$ from both sides,

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k\delta_1\|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \tfrac{L}{2}(1+\mu_k)^2\alpha_k^2\delta_2^2\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.58}$$

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k(\delta_1 + \tfrac{L}{2}(1+\mu_k)\alpha_k\delta_2^2)\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.59}$$

$$\mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)] \leq E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*) - (1+\mu_k)\alpha_k\frac{\delta_1}{2}\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.60}$$

For any vector $\mathbf{p} \in \mathbb{R}^d$, from optimality condition we have,

$$E(\mathbf{p}) \geq E(\hat{\mathbf{w}}_k) + \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}}(\mathbf{p} - \hat{\mathbf{w}}_k) + \tfrac{\rho}{2}\|\mathbf{p} - \hat{\mathbf{w}}_k\|^2 \tag{5.61}$$

$$\geq E(\hat{\mathbf{w}}_k) + \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}}(-\tfrac{1}{\rho}\nabla E(\hat{\mathbf{w}}_k)) + \tfrac{\rho}{2}\|\tfrac{1}{\rho}\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.62}$$

$$\geq E(\hat{\mathbf{w}}_k) - \tfrac{1}{\rho}\|\nabla E(\hat{\mathbf{w}}_k)\|^2 + \tfrac{\rho}{2}\|\tfrac{1}{\rho}\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.63}$$

$$\geq E(\hat{\mathbf{w}}_k) - \tfrac{1}{2\rho}\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.64}$$

Setting $\mathbf{p} = \mathbf{w}^*$,

$$E(\mathbf{w}^*) \geq E(\hat{\mathbf{w}}_k) - \tfrac{1}{2\rho}\|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.65}$$

$$2\rho[E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*)] \leq \|\nabla E(\hat{\mathbf{w}}_k)\|^2 \tag{5.66}$$

We define $\phi_k = \mathbb{E}_{X_k}[E(\hat{\mathbf{w}}_k) - E(\mathbf{w}^*)]$

$$\phi_{k+1} \leq \phi_k - \rho\phi_k(1+\mu_k)\alpha_k\delta_1 \tag{5.67}$$

$$\phi_{k+1} \leq \phi_k(1 - \rho(1+\mu_k)\alpha_k\delta_1) \tag{5.68}$$

By recursive application of the above inequality, we get,

$$\phi_{k+1} \leq \phi_0\left[1 - \rho(1+\mu_k)\alpha_k\delta_1\right]^k \tag{5.69}$$

$$E(\hat{\mathbf{w}}_{k+1})] - E(\mathbf{w}^*) \leq [1 - \alpha(1+\mu_k)\delta_1\rho]^k E(\hat{\mathbf{w}}_0) - E(\mathbf{w}^*) \tag{5.70}$$

The convergence rate is given as

$$\frac{E(\hat{\mathbf{w}}_{k+1}) - E(\mathbf{w}^*)}{E(\hat{\mathbf{w}}_0) - E(\mathbf{w}^*)} \leq [1 - \alpha(1+\mu_k)\delta_1\rho]^k \tag{5.71}$$

Thus Theorem 5.4.1 shows that the aSNAQ algorithm converges to a stationary point $\mathbf{w}^*$ at a linear rate.

□

5.5 Computational Cost

The computation cost is given in Table 5.1. Typical second-order methods such as the BFGS method incur a cost of $nd + d^2 + \zeta nd$ in gradient, Hessian and line-search computation respectively, where $n = |T_r|$ is the number of training samples and d is the number of parameters. In case of NAQ, an additional nd cost is incurred due to twice gradient computation. adaQN and the proposed aSNAQ being stochastic methods, the computation cost in gradient calculation is bd where b is the minibatch size and d is the number of parameters. The Hessian approximation is carried out using the aFIM and two-loop recursion, thus reducing the computation cost to $(4m_L + m_F + 2)d$. However, the error control check adds to an additional cost of $(b + 4)d/L$. aSNAQ has an additional cost bd and d due to twice gradient computation and direction normalization. Furthermore, the storage cost of BFGS and NAQ is d^2 while adaQN and aSNAQ is $(2m_L + m_F)d$. Overall, the cost of both adaQN and aSNAQ are of the order $O(d)$ complexity and hence comparable to that of first-order methods.

Table 5.1: Summary of Computational and Storage Cost.

Algorithm	Computational Cost	Storage
BFGS	$nd + d^2 + \zeta nd$	d^2
NAQ	$2nd + d^2 + \zeta nd$	d^2
adaQN	$bd + (4m_L + m_F + 2)d + (b + 4)d/L$	$(2m_L + m_F)d$
aSNAQ	$2bd + (4m_L + m_F + 3)d + (b + 4)d/L$	$(2m_L + m_F)d$

5.6 Simulation Results

In this section, we evaluate the performance of the proposed method on benchmark problems. The results of the proposed aSNAQ algorithm are compared with first-order Adagrad [12], Adam [22] and second-order adaQN [71] algorithms. The simulations are performed using Tensorflow. For all simulations, we choose the aFIM buffer F size as $m_F = 100$ and the limited memory size for the curvature pairs as $m_L = 10$. The update frequency is chosen to be $L = 5$, learning rate $\alpha = 0.01$ and $\gamma = 1.01$. The momentum update factor ϕ is set to 1.1. All weights are initialized with random normal distribution with zero mean and 0.01 standard deviation. The activation function used is tanh. The hyperparameters of Adagrad and Adam were set to their default values. The performance metrics used for evaluation are accuracy and error. Accuracy is evaluated as a percentage of the number of correct predictions by the RNN compared to the expected output while the error is evaluated by the error function defined for the problem. For all simulations, softmax cross entropy error function is used.

5.6.1 Sequence Counting Problem

The performance of the proposed method is first evaluated on a toy example problem of sequence counting. Given a binary string (a string with just 0s and 1s) of length T, the task is to determine the count of 1s in the binary string. A simple one layer RNN with 24 hidden neurons is chosen. The batch size is set to $b = 50$, sequence length $T = 20$, $\mu_{min} = 0.1$ and $\mu_{max} = 0.99$. Figure 5.2 shows the mean squared error (MSE) over 75 epochs. It can be observed that the proposed method clearly outperforms

adaQN and Adagrad. On comparison with Adam, aSNAQ is faster in the initial iterations and becomes gradually close to Adam.

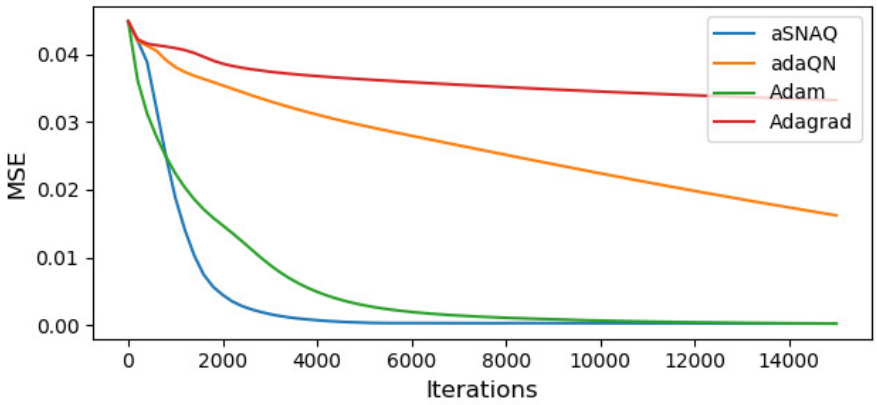
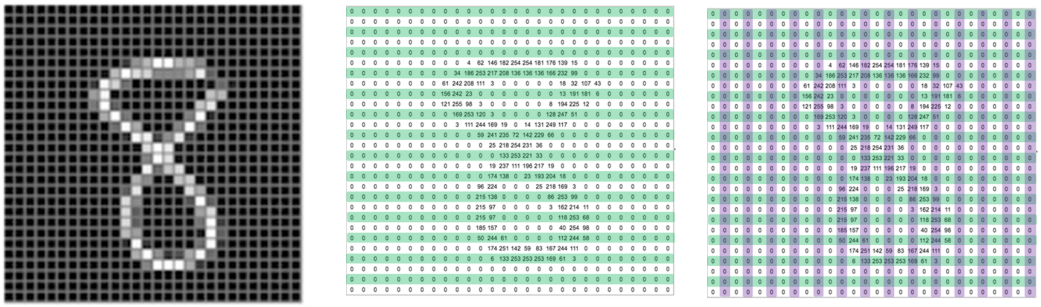


Figure 5.2: MSE for sequence counting problem.

5.6.2 Image Classification

RNNs can be used in image classification problems as well. Since RNNs require a sequence input, for image classification problems, the image is broken into a sequence of pixel values. There are two ways in sequencing images – row-by-row sequence and pixel-by-pixel sequence. In row-by-row sequencing, at each timestep one row is fed as input to the RNN while in pixel-by-pixel sequencing, at each timestep one pixel value is fed as input to the RNN in scanline order starting from the top left to the bottom right pixel.



(a) MNIST pixels (b) Row-by-row sequencing (c) Pixel-by-pixel sequencing

Figure 5.3: Sequencing of the 28×28 pixel MNIST dataset for RNNs

Results on 28×28 MNIST Row by Row Sequence

We study the performance of the proposed algorithm on the standard MNIST image [57] classification problem. The input to the RNN is 28 pixels fed row-wise at each time step, with a total of 28 time steps. We choose batch size $b = 128$, $\mu_{min} = 0.1$, $\mu_{max} = 0.99$. A single layer RNN with 100 hidden neurons is used. Figure 5.4 shows the training error and accuracy over 35 epochs. As seen from the results, we can observe that Adagrad performs poorly and stagnates close to its initial error value. On

comparing the results with Adam and adaQN, it can be noted that aSNAQ performs better than adaQN and is almost on par with Adam.

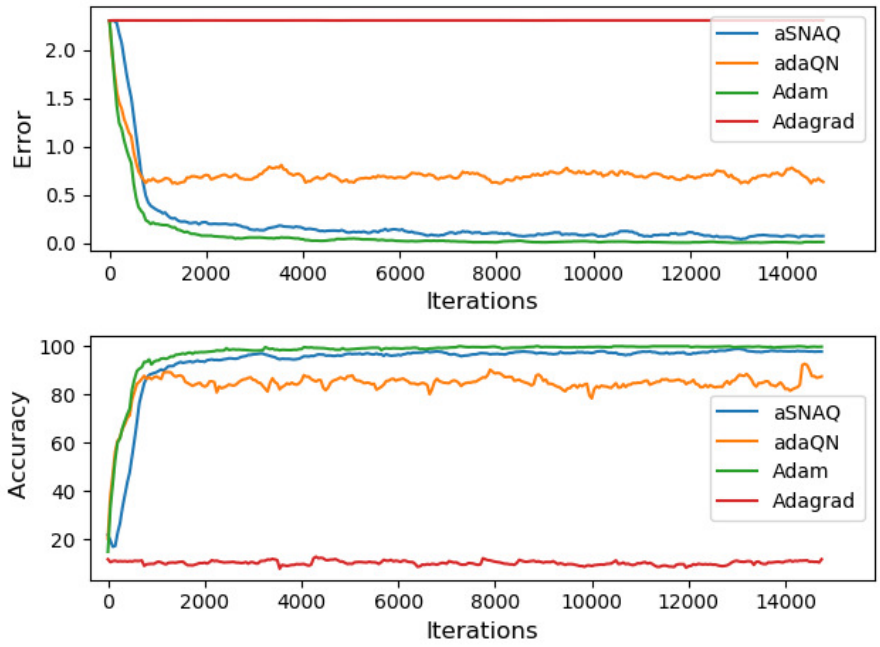


Figure 5.4: Error and accuracy for 28×28 MNIST row by row sequence on training data.

Results on 28×28 MNIST Pixel by Pixel Sequence

We further extend to study the performance of the proposed algorithm on pixel-by-pixel sequential MNIST. The pixel-by-pixel sequence based classification is a challenging task where the 784 pixels are fed to the RNN sequentially in scanline order. Since it involves 784 time steps, it is a long range dependency problem and is much harder compared to the regular classification methods as they are processed one pixel per time step. Though pixel by pixel sequence training is not a conventional approach used for image classification, we merely use this example to illustrate the effects of training long sequences on RNNs. Thus we evaluate the performance on a simple one layer RNN with 100 hidden neurons. We choose batch size $b = 128$, $\mu_{min} = 0.1$, $\mu_{max} = 0.99$. Figure 5.5 shows the training error and accuracy over 35 epochs. In pixel by pixel sequence, both the first order methods Adam and Adagrad methods perform poorly. Though the overall training accuracies are low, aSNAQ shows significant improvement in training compared to adaQN, Adam and Adagrad.

5.6.3 Character Level Language modeling

RNNs are widely used in a number of natural language processing tasks. In this example, we evaluate the performance of the proposed aSNAQ method on character level language modeling problem. The dataset used is The Tale of Two Cities by Charles Dickens. The dataset contains 757,222 characters, split into 80%-20% for train and test samples. The batch size is set to 0.5% of the training set. The vocabulary size, i.e. the number of unique characters including numbers and special characters was 83. The sequence length is set to 50. A 5-layer RNN network with 100 nodes in each layer was used.

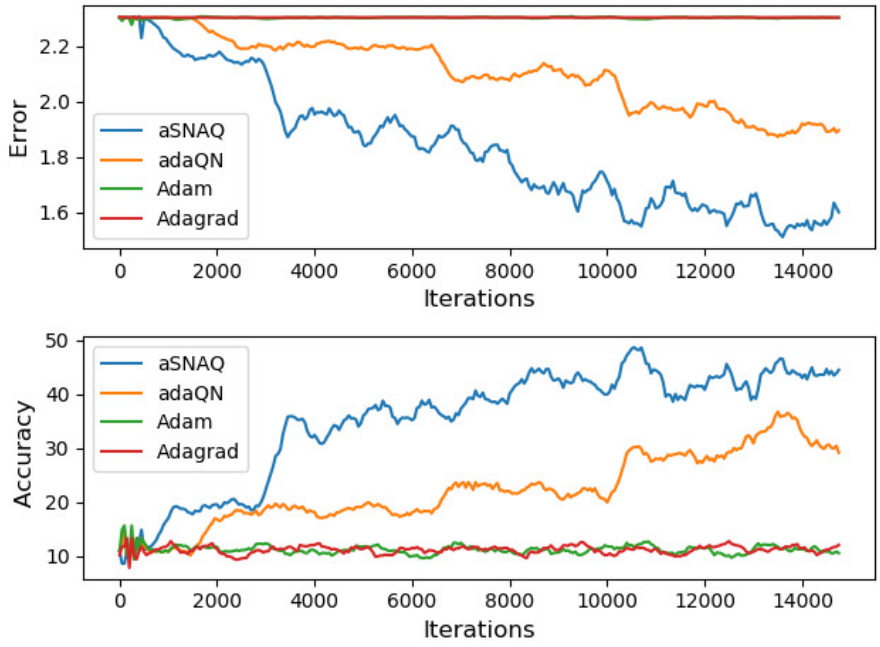


Figure 5.5: Error and accuracy for 28×28 MNIST pixel by pixel sequence on training data.

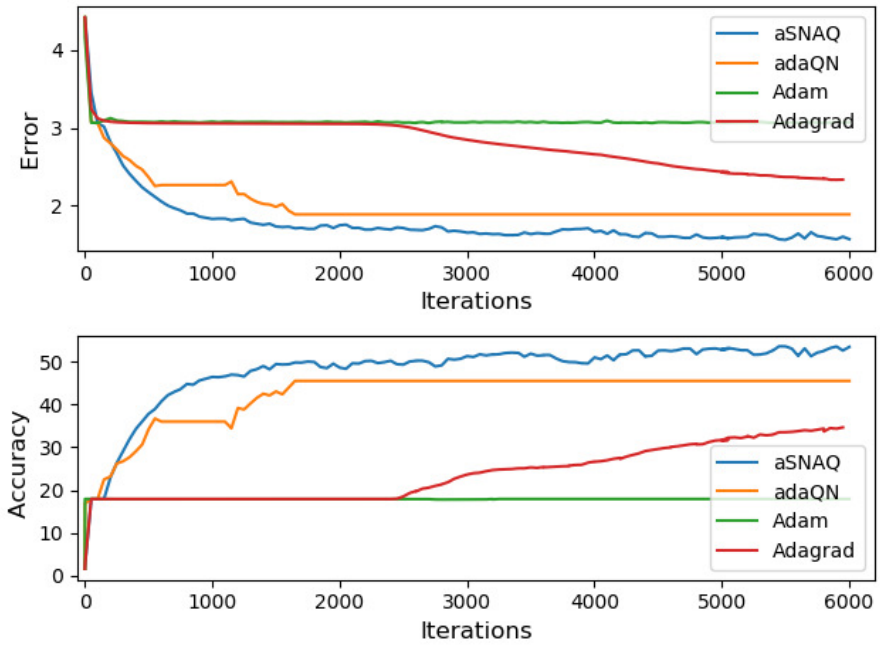


Figure 5.6: Error and accuracy for Character Level Language modeling (5-layer RNN) on test data.

Figure 5.6 shows the error and accuracy of the character level modeling over 30 epochs. From the graph it can be observed that the second-order adaQN and aSNAQ methods perform better compared to the first-order Adagrad and Adam. Further, aSNAQ shows significant acceleration compared to adaQN, thus confirming that the proposed aSNAQ method is suitable for training RNNs at a much faster rate.

5.6.4 Performance on LSTM

To further validate the performance of the proposed method on other RNN architectures, we consider the character level language modeling problem using LSTM architecture. LSTMs (Long short term memory) [67] were especially designed to model long range sequences with long-term dependencies by introducing gate units. A typical LSTM cell consists of an input gate, forget gate and output gate, that filters the information being passed, and thus help in solving the vanishing and exploding gradient issue. In this problem, we consider a two layer LSTM network with 100 hidden neurons each. Same as

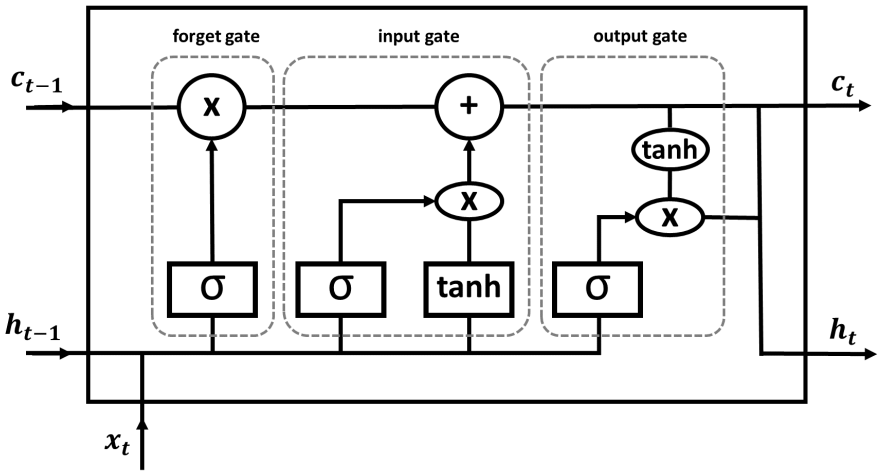


Figure 5.7: Structure of a long-short term memory (LSTM) unit.

in section 4.3, the dataset used is The Tale of Two Cities by Charles Dickens. The dataset is split into 80%-20% for train and test samples and the batch size is set to 0.5% of the training set. The vocabulary size is 83 and the sequence length is set to 50. Figure 5.8 shows the error and accuracy over 30 epochs. The results obtained confirm that the proposed aSNAQ algorithm can be effectively used even for other RNN architectures. It can be observed that in this problem Adam performs the best. However, aSNAQ still performs better than Adagrad and adaQN. Further improvement in the overall performance of the model can be achieved by increasing the model depth.

5.7 Discussion

From the above simulation results, it can be observed that the proposed aSNAQ method shows good performance with sufficiently small errors and good accuracies compared to the second-order adaQN and first-order Adagrad and Adam methods. RNNs commonly used in modeling long sequences with long-term dependencies are difficult to train due to the vanishing and exploding gradient issue. Both, the first and second-order methods were able to train a vanilla RNN for a simple toy example such as the sequence counting problem. For examples with short dependencies (section 4.1 and 4.2.1), Adam performed the best with errors smaller than aSNAQ. However, as the sequence length or the scale of the network increases, the first-order methods do not perform well on vanilla RNN. On the other hand, both the second-order methods – adaQN and aSNAQ were consistent and effective in training the RNN. For the same language modeling problem described in section 5.6.3 and 5.6.4, Adam

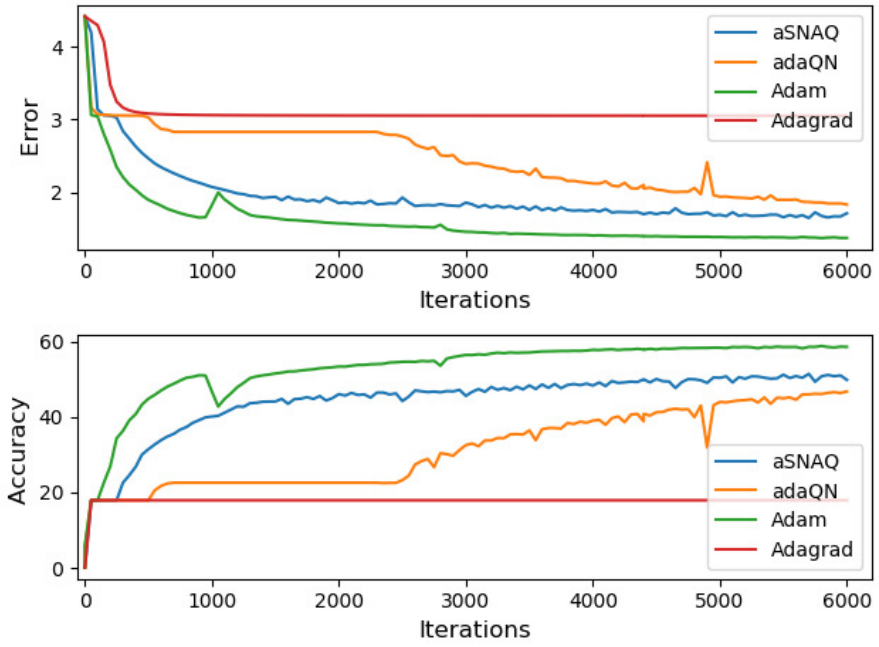


Figure 5.8: Error and accuracy for Character Level Language modeling on 2-layer LSTM network on test data.

performed the best on the 2-layer LSTM network, whereas on the 5-layer vanilla RNN it performed the worst. This could be possibly because LSTMs are resilient to the vanishing and exploding gradient problem. However, in all the examples, aSNAQ showed better performance compared to adaQN with significantly higher accuracies and low errors. Also, the direction normalization implemented in aSNAQ is similar to gradient clipping, which is a popular solution to the vanishing and exploding gradient issue. Hence, it can be deduced from the consistent performance of aSNAQ that it is efficient in combating the vanishing and exploding gradient issue. On comparing the computation cost, it can be noted that adaQN and aSNAQ have a low per-iteration cost of the order $O(d)$ since $m_L, m_F \ll d$ and is comparable in terms of the order of computational complexity of first-order methods. Though aSNAQ has a slightly higher computation cost compared to adaQN, it is well compensated by its accelerated performance. Furthermore, aSNAQ shows to be effective not only on vanilla RNNs, but also LSTM and hence suggest a good feasibility to be applied to different problems and architectures. A detailed analysis of the proposed method on more examples and different network structures should be performed to further validate the effectiveness of the proposed aSNAQ algorithm.

5.8 Summary

In this chapter we discussed on the proposed adaptive stochastic Nesterov's accelerated quasi-Newton method for training recurrent neural networks (RNNs). The proposed aSNAQ method is an accelerated method that combines adaQN with NAQ by introducing the Nesterov's accelerated gradient and momentum term. aSNAQ attempts to incorporate second-order curvature information while maintaining a low per-iteration cost which is of the order $O(d)$. The performance of the proposed method was verified on benchmark image classification and language modeling problems. From the simulation

results, it was confirmed that incorporating the Nesterov's accelerated gradient and momentum term improves the performance in the training of RNNs compared to adaQN and other popular first-order methods such as Adagrad and Adam. Further evaluation on LSTM network validates the effectiveness of aSNAQ on other architectures as well. We further provide the convergence analysis of the proposed algorithm and show that aSNAQ converges to a stationary point at a linear rate.

6

Quasi-Newton Methods for Deep Reinforcement Learning

Recent advances in deep reinforcement learning has led to its application in a number of real-world problems. One of the most popularly used deep reinforcement learning algorithms is the deep Q-learning method which uses neural networks to approximate the estimation of the action-value function. Training of deep Q-networks (DQN) is usually restricted to first-order gradient based methods. Though second-order methods have shown to have faster convergence in several supervised learning problems, their application in deep reinforcement learning is limited. This chapter demonstrates the efficiency of our proposed aSNAQ method, in accelerating the training of deep Q-networks. As a use case for application in real-world examples, we consider the VLSI global routing problem, which is modeled using deep reinforcement learning to obtain optimum routing solutions. This chapter is based on results published in [74] and [75].

6.1 Introduction

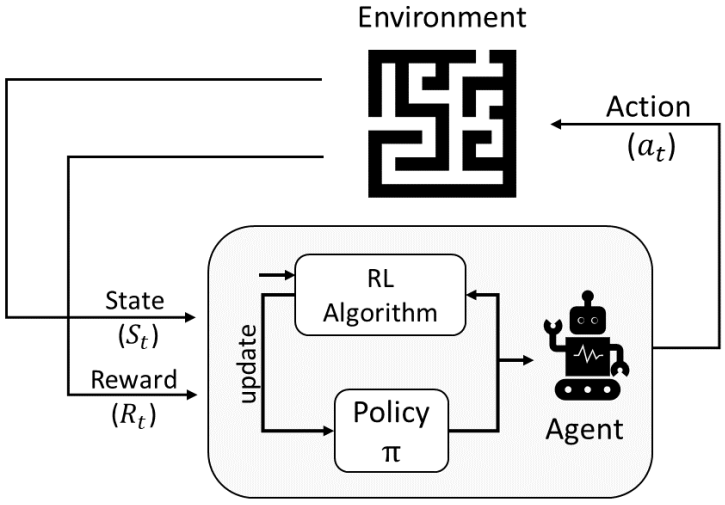


Figure 6.1: Reinforcement learning model (Reproduced from [76]).

Reinforcement learning (RL) is a machine learning technique where an agent perceives its current state and takes some actions by interacting with an environment [76]. At each time step t, the agent receives an observation of the current state s of the enviroment, based on which the agent chooses an action a, for which the environment returns a reward r and updates its current as s'. Each cycle of this agent-environment interaction is called an experience $\mathcal{M} = \{s, a, r, s'\}$ which is stored in memory for training the reinforcement learning algorithm. The reinforcement learning algorithm attempts to find a policy π, the function mapping from states to actions, for maximizing the cumulative reward $\mathcal{R}$ over the course of the problem. Figure 6.1 shows a pictorial representation of a reinforcement learning model.

The Q-learning algorithm [77] is one of the popular off-policy reinforcement learning algorithms that chooses the best action based on estimates of the state-action value $Q(s, a)$ represented in the form of a table called the Q-table. As the state and action space of the problem increases, the estimation of the state-action value can be slow and time consuming. Hence, the state-action value (Q-value) is often estimated as a function approximation. These function approximations can be represented as a non-convex, non-linear unconstrained optimization problem and can be solved using deep neural networks (known as deep Q-networks) [78]. Training neural networks in reinforcement learning tasks is usually slow and challenging due to the training data being temporally correlated, non-stationary and presented as a stream of experiences rather than batches like in supervised learning. Training of Deep Q-Networks (DQN) are usually restricted to first-order methods such as stochastic gradient descent (SGD), RMSprop [21], Adam [22], etc. Using second-order curvature information have shown to improve the performance and convergence speed for non-convex optimization problems. In extension to the previous chapter which showed the efficiency of the adaptive Stochastic Nesterov's Accelerated Quasi-Newton method (aSNAQ), this chapter attempts to verify the feasibility aSNAQ in training deep Q-Networks, considering that this type of training is more challenging compared to batch training since the training samples in reinforcement learning are a continuous stream of experiences, that makes it more prone to unlearning effective features over time. The evaluation shows that aSNAQ allows more stable approximations and is efficient in training DQNs for deep reinforcement learning applications.

The rest of the chapter is organized as follows: section 6.2 gives a brief background of reinforcement learning and its notations, followed by an introduction to the aSNAQ algorithm for DQN in section 6.3. We study the performance of the proposed method in an example of solving the VLSI global routing problem using deep reinformcement learning, the framework of which is detailed in section 6.4. In section 6.5, we summarize the results and conclude in section 6.6.

6.2 Background

The Reinforcement Learning (RL) problem is typically modeled as a Markov's Decision Process (MDP). In order to solve the MDP, the estimates of the value function of all possible actions is learnt using Q-learning method, a form of temporal difference learning [77]. The Q-learning algorithm estimates the state-action value $Q(s, a)$ for all possible state and action combinations, represented in the form of a table called the Q-table, and the action corresponding to the largest state-action value is chosen in the view of maximizing the cumulative future reward. With increase in the state-action space of the problem, the estimation of the state-action values become complex and is thus estimated by function approximations which are then solved by deep neural networks, known as deep Q-networks

(DQNs) [78]. The optimal action-value function $Q^*(s, a)$ that maximizes the cumulative reward satisfies the Bellman equation and is given as

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{M}}[\mathcal{R} + \gamma \max_{a'} Q^*(s', a')|s, a], \tag{6.1}$$

where γ is the discount factor. The Q-learning algorithm is an off-policy, model-free reinforcement learning algorithm that iteratively learns the optimal action-value function. In deep Q-learning, this function is optimized by a neural network parameterized by $\mathbf{w}$. The inputs to the neural network are the states s and the output predicted by the neural network correspond to the action values $Q(s, a; \mathbf{w})$ for each action a. A replay buffer of fixed memory $\mathcal{M}$ stores the transitions (s, a, r, s') of the agent's experiences from which samples are randomly drawn for training the DQN. The loss function $\mathcal{L}(\mathbf{w})$ as shown in (6.2) is used in backpropagation and calculation of the gradients for updating the parameters $\mathbf{w}$.

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{(s,a)\sim\mathcal{M}}[(\mathcal{Y} - Q_{\mathbf{w}}(s, a))^2], \tag{6.2}$$

where the target function $\mathcal{Y}$ is given as

$$\mathcal{Y} = \mathbb{E}_{(s')\sim\mathcal{M}}[r + \gamma \max_{a'} Q_{\mathbf{w}}(s', a')]. \tag{6.3}$$

DQNs are said to overestimate the update of the action-value function since $Q_{\mathbf{w}}(s, a)$ is used to select the best next action at state s' and apply the action value predicted by the same $Q_{\mathbf{w}}(s, a)$. Double deep Q-learning [79] resolves this issue by decoupling the action selection and action value estimation using two Q-networks. The target function $\mathcal{Y}$ of double DQN is given as

$$\mathcal{Y} = \mathbb{E}_{(s')\sim\mathcal{M}}[r + \gamma Q_{\mathbf{w}^-}(s', \text{argmax}_{a'} Q_{\mathbf{w}}(s', a'))]. \tag{6.4}$$

$\mathbf{w}$ and $\mathbf{w}^-$ represent the parameters of the two Q networks – primary and target networks, respectively. The primary network parameters are periodically copied to the target network using Polyak averaging as shown in (6.5), where τ is a small value such as 0.05.

$$\mathbf{w}^- \leftarrow \tau\mathbf{w} + (1 - \tau)\mathbf{w}^-. \tag{6.5}$$

At each iteration of the training, the parameters of the DQN are updated as

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_k, \tag{6.6}$$

where $\mathbf{v}_k$ is the update vector which varies for each training algorithm. The update vector for first-order methods take the form

$$\mathbf{v}_k = -\alpha\nabla\mathcal{L}(\mathbf{w}_k), \tag{6.7}$$

where $\nabla\mathcal{L}(\mathbf{w}_k)$ is the gradient of the loss function calculated on a small mini-batch sample drawn at random from the experience replay buffer $\mathcal{D}$.

6.3 Nesterov's Accelerated Quasi-Newton Method for Q-learning

First order gradient based methods have been commonly used in training DQNs due to their simplicity and low computational complexity. However approximated second-order quasi-Newton methods have

Algorithm 6.1 aSNAQ for DQN

Require: minibatch X_k, μ_{min}, μ_{max}, k_{max}, $\mathcal{E}_{max}$, aFIM buffer $\mathbf{F}$ of size m_F and curvature pair buffer $(\mathbf{S}, \mathbf{Y})$ of size m_L, momentum update factor ϕ, experience replay buffer $\mathcal{M}$

Initialize: $\mathbf{w}_o$=$\mathbf{w}_k \in \mathbb{R}^d$, $\mu = \mu_{min}$, $\mathbf{v}_k$, $\mathbf{v}_o$, $\mathbf{w}_s$, $\mathbf{v}_s$, and $t = 0$

for episode $\mathcal{E} = 1, 2, ..., \mathcal{E}_{max}$ **do**
  Initialize state s
  for step $k = 1, 2, ..., k_{max}$ **do**
    Take action a based on ϵ-greedy strategy
    Store transistion (s, a, r, s', a') in $\mathcal{M}$
    Sample random minibatch X_k from $\mathcal{M}$
    Calculate $\nabla\mathcal{L}(\mathbf{w}_k + \mu\mathbf{v}_k)$
    Determine $\mathbf{g}_k$ using two loop recursion (Appendix A.1)
    $\mathbf{g}_k = \mathbf{g}_k / \|\mathbf{g}_k\|_2$
    $\mathbf{v}_{k+1} \leftarrow \mu\mathbf{v}_k + \alpha_k\mathbf{g}_k$
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \mathbf{v}_{k+1}$
    Calculate $\nabla\mathcal{L}(\mathbf{w}_{k+1})$ and store in $\mathbf{F}$
    $\mathbf{w}_s = \mathbf{w}_s + \mathbf{w}_{k+1}$ and $\mathbf{v}_s = \mathbf{v}_s + \mathbf{v}_{k+1}$
    if $\mathrm{mod}(k, L) = 0$ **then**
      Compute avg $\mathbf{w}_t = \mathbf{w}_s / L$ and $\mathbf{v}_t = \mathbf{v}_s / L$
      $\mathbf{w}_s = \mathbf{0}$ and $\mathbf{v}_s = \mathbf{0}$
      if $t > 0$ **then**
        if $\mathcal{L}(\mathbf{w}_t) > \eta\mathcal{L}(\mathbf{w}_{t-1})$ **then**
          Clear $(\mathbf{S}, \mathbf{Y})$ and $\mathbf{F}$ buffers
          Reset $\mathbf{w}_k = \mathbf{w}_{t-1}$ and $\mathbf{v}_k = \mathbf{v}_{t-1}$
          Update $\mu = \max(\mu/\phi, \mu_{min})$
          continue
        end if
        $\mathbf{s} = \mathbf{w}_t - (\mathbf{w}_{t-1} + \mu\mathbf{v}_{t-1})$
        $\mathbf{y} = \frac{1}{|\mathbf{F}|}(\sum_{i=1}^{|\mathbf{F}|} \mathbf{F}_i \cdot \mathbf{s})$
        Update $\mu = \min(\mu \cdot \phi, \mu_{max})$
        if $\mathbf{s}^{\mathrm{T}}\mathbf{y} > \sigma\ \mathbf{s}^{\mathrm{T}}\mathbf{s}$ **then**
          Store curvature pairs (**s**,**y**) in $(\mathbf{S}, \mathbf{Y})$
        end if
      end if
      $t \leftarrow t + 1$
    end if
  end for
end for

shown to signficantly speed up convergence in non-convex optimization problems. The Nesterov's accelerated quasi-Newton (NAQ) method [31] and its variants [32, 37] have shown to accelerate convergence compared to the standard quasi-Newton method in various supervised learning frameworks. Reinforcement learning is different from traditional supervised learning in terms that the training data distribution changes as the policy improves, which results in the objective function being non-stationary [80]. The performance of deep reinforcement learning is thus sensitive to hyperparameters, choice of architecture [81], replay buffer size and optimizer hyperparameters [82]. This study extends our previous work [64,75] by investigating the feasibility of our proposed adaptive stochastic Nesterov's Accelerated Quasi-Newton (aSNAQ) algorithm in training deep Q-networks and show that the aSNAQ

method allows more stable approximations for deep reinforcement learning applications. The aSNAQ algorithm for training DQN is shown in Algorithm 6.1.

The Nesterov's accelerated quasi-Newton (NAQ) method achieves faster convergence by quadratic approximation of the objective function at $\mathbf{w}_k + \mu\mathbf{v}_k$ and by incorporating the Nesterov's accelerated gradient $\nabla\mathcal{L}(\mathbf{w}_k + \mu\mathbf{v}_k)$. The aSNAQ algorithm builds on the algorithmic framework of adaQN [71] and NAQ [31]. The update vector $\mathbf{v}_k$ can be written as

$$\mathbf{v}_{k+1} = \mu\mathbf{v}_k + \alpha_k\mathbf{g}_k, \tag{6.8}$$

where $\mathbf{g}_k$ is the search direction given by

$$\mathbf{g}_k = -\mathbf{H}_k\nabla\mathcal{L}(\mathbf{w}_k + \mu\mathbf{v}_k). \tag{6.9}$$

The search direction vector $\mathbf{g}_k$ is computed using the two-loop recursion [1] as shown in Appendix A.1. The Hessian $\mathbf{H}_k$ is initialized with non-constant diagonal entries based on the accumulated gradient information as

$$[\mathbf{H}_k^{(0)}]_{ii} = \frac{1}{\sqrt{\sum_{j=0}^{k}\nabla\mathcal{L}(\mathbf{w}_j)_i^2 + \epsilon}}, \tag{6.10}$$

where $\epsilon = 10^{-4}$ is used to prevent numerical instability. aSNAQ uses an accumulated Fisher Information Matrix (aFIM) for computing the curvature information pair $(\mathbf{s}, \mathbf{y})$ for the Hessian computation as shown in (6.11) and (6.12)

$$\mathbf{s} = \mathbf{w}_t - (\mathbf{w}_{t-1} + \mu\mathbf{v}_{t-1}), \tag{6.11}$$

$$\mathbf{y} = \frac{1}{|\mathbf{F}|}\sum_{i=1}^{|\mathbf{F}|}\mathbf{F}_i \cdot \mathbf{s}\,, \tag{6.12}$$

where $\mathbf{w}_t$ is the average aggregated weight, t is the update counter of the curvature information pair, $\mathbf{F}_i = \nabla\mathcal{L}(\mathbf{w}_{k+1})\nabla\mathcal{L}(\mathbf{w}_{k+1})^{\mathrm{T}}$ and $|\mathbf{F}|$ is the number of entries present in $\mathbf{F}$. The $\mathbf{y}$ vector is computed without explicitly constructing the $\nabla\mathcal{L}(\mathbf{w}_{k+1})\nabla\mathcal{L}(\mathbf{w}_{k+1})^{\mathrm{T}}$ matrix by just storing the $\nabla\mathcal{L}(\mathbf{w}_{k+1})$ vector. The use of the Fisher Information matrix (aFIM) gives a better estimate of the curvature of the problem. In deep Q-learning, the estimated stochastic gradient has a large variance as it is estimated on the basis of the data collected from only one step action or a small batch of actions. Thus it often suffers from inaccurate estimation of the gradients which can be viewed as distorted gradient direction [8, 83]. The curvature information pair $(\mathbf{s}, \mathbf{y})$ is computed based on the average of the weight aggregates and Hessian-vector product, and thus reduces the effect of noise and allows for more stable approximations. Furthermore, normalizing the search direction at each iteration ensures that the algorithm does not move too far away from the current objective [49]. The curvature pairs are computed every L steps and stored in the $(\mathbf{S}, \mathbf{Y})$ buffer only if sufficiently large. This allows for the updates being made only based on useful curvature information. The size of the $(\mathbf{S}, \mathbf{Y})$ buffer and aFIM buffer F are set to m_L and m_F, respectively, thus optimizing storage cost to $(2m_L + m_F)d$ compared to a d^2 storage cost incurred in BFGS and NAQ, where d is the number of parameters. The control heuristics as shown in steps 18-22 further allows to take a step back to a previously learnt parameter $\mathbf{w}$ if the loss $\nabla\mathcal{L}(\mathbf{w}_t)$ is larger compared to the loss $\nabla\mathcal{L}(\mathbf{w}_{t-1})$. Though this incurs an additional cost of $(b+4)d/L$, where b is the batch size, the control heuristics contributes to better approximations and thus early convergence. The momentum parameter μ is also adaptively updated as shown in steps 21 and 26. Further, the

curvature pair information $(\mathbf{s}, \mathbf{y})$ and hence the Hessian approximation is updated once in L iterations, thus reducing computational cost. The order of computational complexity of second-order methods is usually $O(d^2)$ in case of quasi-Newton methods and $O(d^3)$ in case of Newton methods, while aSNAQ reduces the computation cost to $2bd + (4m_L + m_F + 3)d + (b + 4)d/L$, which is in the order of $O(d)$, and comparable to first-order methods.

6.4 VLSI Global Routing

Synthesis and physical design optimizations are the core tasks of the VLSI / ASIC design flow. Global routing has been a challenging problem in VLSI physical design. VLSI physical design requires to compute the best physical layout of millions to billions of circuit components on a tiny silicon surface ($< 5\text{cm}^2$). It is carried out in several stages such as partitioning, floor-planning, placement, routing and timing-closure. In the placement stage, the locations of the circuit components, i.e. cells, are determined. Once all cell locations are set, the paths for all the connections of the circuit, i.e. nets, are determined in the routing stage. Global routing involves a large and arbitrary number of nets to be routed, where each net may consist of many pins to be interconnected with wires. In addition, the IC design consideration may impose several constraints such as number of wire crossings (capacity), blockages or congestion, spacing between wires, etc. In global routing, given a netlist with the description of all the components, their connections and positions, the goal of the router is to determine the path for all the connections without violating the constraints and design rules. Conventional routing automation tools are usually based on analytical and path search algorithms such as A* search [84], which are NP complete. Hence a machine learning approach would be more suitable for this kind of automation problem. Most studies that propose AI techniques such as machine learning, deep learning, genetic algorithms deal with only prediction of routability, short violations, pin-access violations, etc [85, 86]. Moreover, the non-availability of large labelled training datasets for a supervised learning model is another challenge. Thus deep reinforcement learning (DRL) is a potential approach to such applications. Reinforcement learning has been successfully used in several applications ranging from Atari games [87] to controlling of high degree of freedom in robots [88]. Recently, a reinforcement learning approach to global routing that uses first-order gradient based method for training the DQN was proposed in [89]. In this chapter, we attempt to accelerate the training of deep Q-networks by introducing a second-order Nesterov's accelerated quasi-Newton method to get better routing solutions in fewer episodes. Also, to further enhance the performance of the DRL model for global routing, we propose using double deep Q-learning [79]. The obtained routing solution is evaluated in terms of total wirelength and overflow and compared with the results of double DQNs trained using Adam and RMSprop.

6.4.1 Global Routing Modelling

The global routing problem is commonly modeled as a grid graph $G(V, E)$ where V represents a grid tile where the components are placed and E represents an edge along which wires are drawn [90]. Each vertical and horizontal edge composing a grid tile is associated with a capacity to limit the number of wire-crossings. The objective of the router is to find the optimum connections (routing solutions) for all the pins such that the total wirelength is minimum and no overflow occurs. An overflow is said to occur when the number of wire-crossings at an edge (utilization) exceeds the capacity set for that

particular edge. For every wire routed, the corresponding capacity decreases. Routing is sequential and hence a common problem is net ordering. Nets routed early can block the routes for later nets due to utilization of the capacity. Consider an example of a two layer 4×4 grid graph as shown in Figure 6.2. The netlist consists of two pairs of pins, $S1 - G1$ and $S2 - G2$ to be routed. In layer 1, each vertical edge has a capacity of one and each horizontal edge has a capacity of zero while in layer 2 each vertical edge has a capacity of zero and each horizontal edge has a capacity of one. Thus the routing direction on layer 1 is horizontal and layer 2 is vertical. That is, only horizontal wires can be drawn in layer 1 and only vertical wires can be drawn in layer 2 as seen in Figure 6.2. A pair of vias are used to connect between layers. The rectangular blocks represent congestion or blockages and the capacity corresponding to them is zero and hence no wires can pass through. The total wirelength is calculated by summing the total wire segments in each layer and vias between layers. A wirelength of one corresponds to the wire segment passing from one grid tile to another. The solution indicated in this figure consists of 5 horizontal wire segments in layer 1 and 4 vertical wire segments in layer 2 and 4 via pairs connecting layer 1 and layer 2. Therefore the wirelength is 13 and overflow is 0 since the utilization does not exceed the capacity.

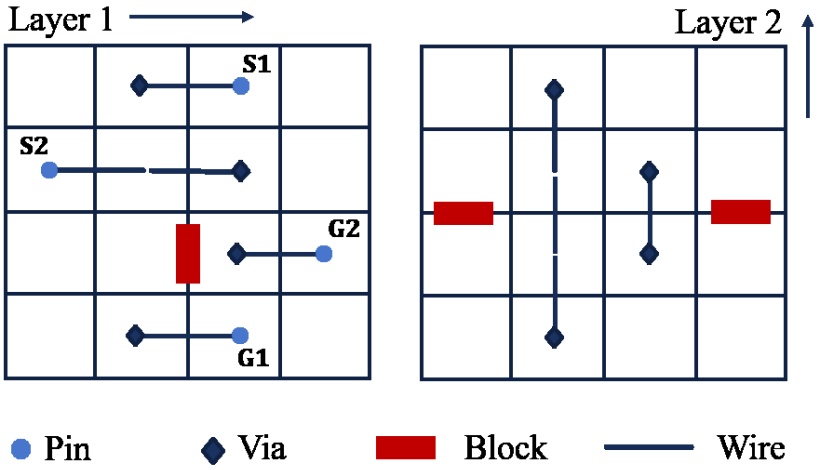


Figure 6.2: Example of a routing solution.

The global routing problem can be modeled as a 3-dimensional grid-world or maze problem with multiple start-goal pairs corresponding to the pins to be routed and be solved by reinforcement learning. The study proposed in [89] shows potential scope for reinforcement learning based global routing, where the DQN is trained using a first-order gradient based algorithm. Here we evaluate the efficiency of our second-order aSNAQ algorithm on the reinforcement learning framework for global routing. In the following section, we briefly explain the adopted reinforcement learning framework for global routing as proposed by [89].

6.4.2 Deep Reinforcement Learning Framework for Global Routing

The global routing problem is solved using a deep reinforcement learning framework in which for each two-pin, the agent interacts with the environment and provides the Q-network with a 12 dimensional state vector $s = \{s_1, s_2 ... s_{12}\}$. The elements s_1, s_2, s_3 correspond to the agent's current position in

terms of x, y, z coordinates; s_4, s_5, s_6 correspond to the distance in x, y, z coordinates from the current position to the target pin position; and $s_7, s_8, ..., s_{12}$ correspond to the capacity information in each of the six directions. The action space consists of 6 possible actions corresponding to the direction in which the agent can move, i.e., up, down, left, right, layer n to layer $n+1$, and layer $n+1$ to layer n. We use a small feedforward neural network to estimate the state-action values $Q(s,a) = \{q_1, q_2, ..., q_6\}$ corresponding to each of the 6 actions and the agent takes an action based on the ϵ-greedy strategy.

$$A(s) = \begin{cases} \text{random action } a & \text{with } \epsilon \text{ probability,} \\ \text{argmax}_a Q_{\mathbf{w}}(s,a) & \text{with } 1-\epsilon \text{ probability.} \end{cases} \tag{6.13}$$

The environment returns a reward $R(a, s')$ and the capacity information is updated.

$$R(a,s') = \begin{cases} +100 & s' \text{ is the target pin,} \\ -1 & \text{otherwise.} \end{cases} \tag{6.14}$$

The experience replay buffer $\mathcal{M}$ stores the transitions $\{s, a, r, s'\}$ which is used for training and updating the weights using backpropagation. The Q-network architecture used is 12–32–64–32–6 with ReLU activation.

6.5 Simulation Results

The performance of our proposed aSNAQ method is evaluated in solving global routing using deep reinforcement learning. In order to further enhance the performance and stability we use double DQN and consider a two layer $n \times n$ grid with two-pins nets. In the following sections, we evaluate the performance of aSNAQ in comparison to popular first-order Adam and RMSprop. The discount factor γ is set to 0.9. A batch size of 32 is chosen. The performance metrics include the total wirelength and overflow. For all successful solutions i.e. all nets routed within the maximum number of episodes, zero overflow was obtained and the corresponding total wirelength was calculated using the ISPD'08 contest evaluator. The routing solution obtained using A* search is set as the baseline for comparison of the performance metrics. The benchmark netlists were generated using the open-sourced problem

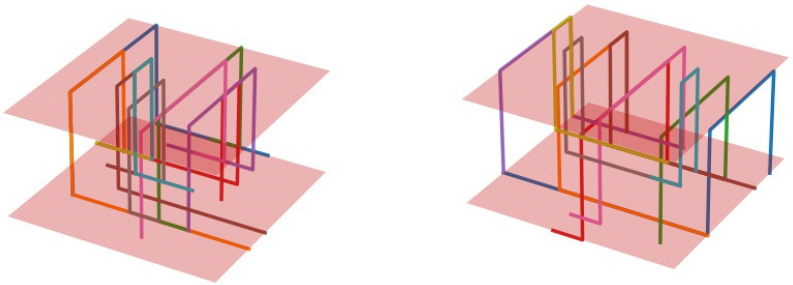
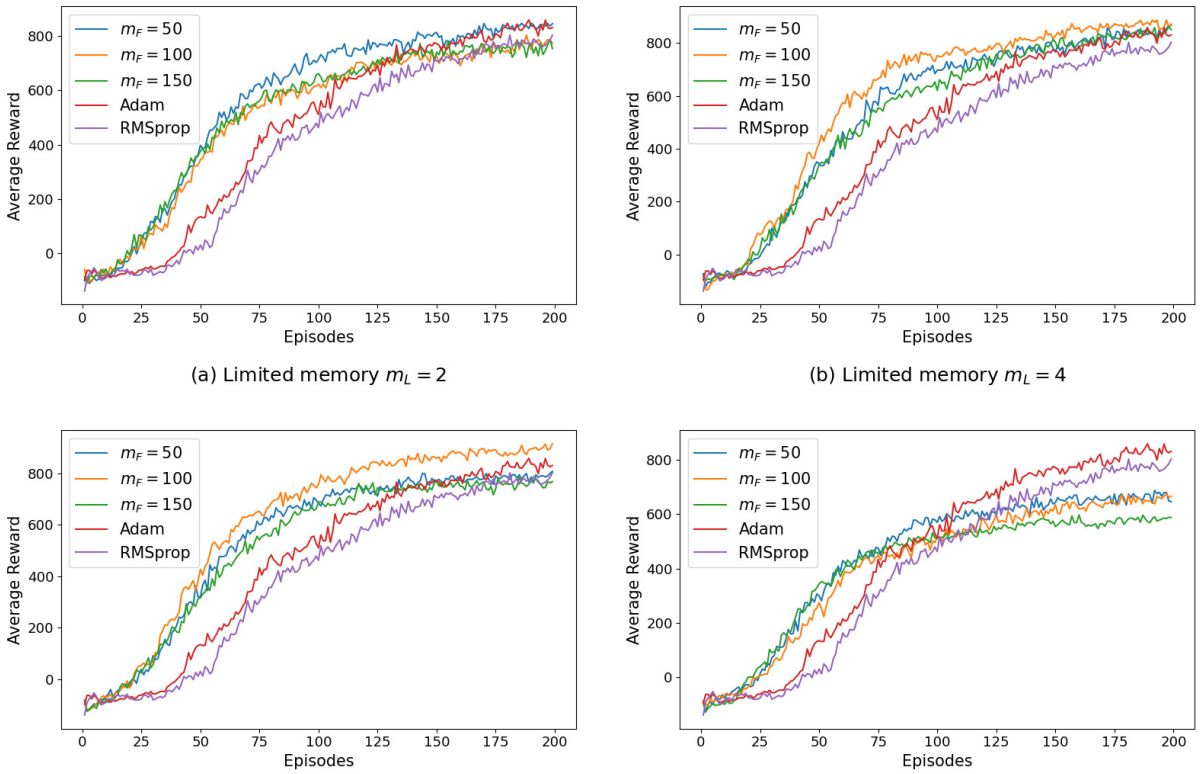


Figure 6.3: Examples of routing of netlists generated by the problem set generator [89].

set generator [89]. The problem set generator can generate a number of netlist files complying to the ISPD'08 input format, with user-specified grid-size, number of nets, number of pins in each net, edge capacity and blockages. Each netlist comprises of a list of several nets where each net is a collection of two or more pins or components. The pin positions are arbitrarily set and thus each netlist file generated is different. Figure 6.3 shows an example of two netlists whose routing solutions were obtained using the A* search method. Each of these netlists consist of 10 nets with 2 pins each.

6.5.1 Discussion on the choice of m_L and m_F

The two main hyperparameters in aSNAQ are the limited memory size m_L and the accumulated Fisher Information Matrix (aFIM) size m_F. We evaluate the performance of aSNAQ for different values of the limited memory size m_L and aFIM size m_F, chosen from the set $m_L = \{2, 4, 8, 16\}$ and $m_F = \{50, 100, 150\}$, respectively. For the evaluation, let us consider a small example of a two layer 5×5 grid with 10 nets with edge capacity 3 and 3 blockages. The edge capacity defines the number of wires that can be drawn across an edge and thus gives a limit on the congestion. Blockages denote edges where no wires can be drawn (i.e. the edge capacity is 0). It denotes those regions of the printed circuit board (PCB) that are reserved for a special purposes, such as a port. We generated a total of 25 benchmarks using the open-sourced problem set generator [89]. We train our reinforcement learning model to route 25 benchmark problems with the maximum number of episodes $\mathcal{E}_{max}$ set to 200, where an episode constitutes a single pass over the entire set of pin pairs over all nets and the maximum steps



(a) Limited memory $m_L = 2$

(b) Limited memory $m_L = 4$

(c) Limited memory $m_L = 8$

(d) Limited memory $m_L = 16$

Figure 6.4: Comparison of average reward for different values of m_L and m_F.

k_{max} for each episode is set to 20.

Figure 6.4 shows the average reward over 25 benchmarks for different combinations of m_L and m_F in comparison to Adam and RMSprop. From the figure, we can observe that for all combinations of m_L and m_F aSNAQ shows higher average reward in the initial episodes compared to both Adam and RMSprop. This confirms the acceleration of aSNAQ in training the DQNs. However, for $m_L = 16$ it can be observed from Figure 6.4(d) that aSNAQ does not perform well after 75 episodes. From Figure 6.4(b) we can observe that for $m_L = 4$, all three values of m_F are almost on par with Adam and better than RMSprop. For $m_L = 2$, a aFIM size of $m_F = 50$ performs the best and for $m_L = 8$, aFIM of size $m_F = 100$ performs the best.

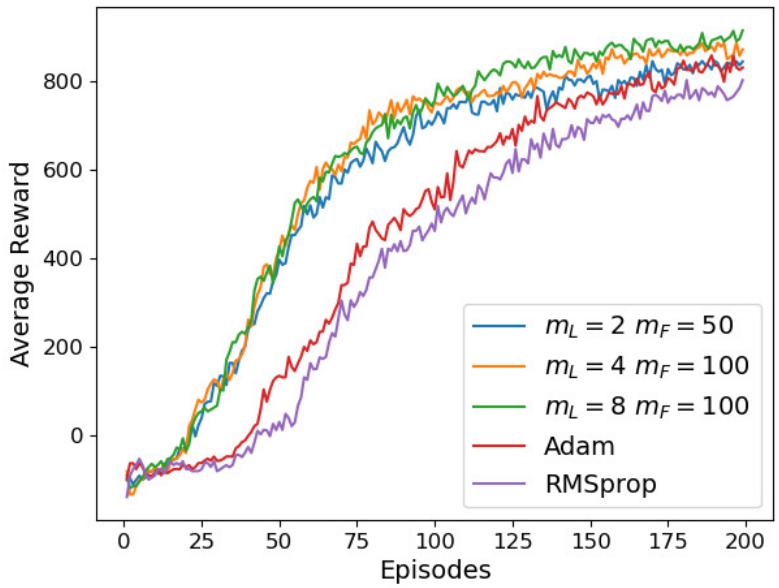


Figure 6.5: Average reward over 25 benchmarks with 10 two-pin nets.

Figure 6.5 shows the comparison of aSNAQ with the best m_F corresponding to $m_L = 2, 4$ and 8. From Figure 6.5, it can be observed that $m_F = 100$ is better compared to $m_F = 50$. While both $m_L = 4$ and $m_L = 8$ with $m_F = 100$ are almost on par with each other, $m_L = 8$ is comparitively better than $m_L = 4$. For all successful solutions i.e. all nets routed within the maximum number of episodes $\mathcal{E}_{max} = 200$, zero overflow was obtained and the corresponding total wirelength was calculated using the ISPD'08 contest evaluator. Adam could route 20 out of 25 benchmarks, RMSprop could route 15 of 25 benchamrks and $m_L = 4$ with $m_F = 100$ could route 22 out 25 benchmarks while $m_L = 8$ with $m_F = 100$ could route 23 out of 25 benchmarks. Since the example considered is small, on comparing the routing solution obtained from the DRL trained with aSNAQ, there was no further reduction in wirelength compared to the A* search solution.

6.5.2 Performance comparison of aSNAQ in routing 50 nets

Next, let us consider a two-layer 8×8 grid with a total of 50 nets with two pins in each net. The capacity of the edges are set to 5 and the number of blockages is set to 3. We generated a total of 30 benchmarks with the above settings using the open-sourced problem set generator [89]. Since the grid

size is larger compared to the previous example, we set the maximum number of episodes $\mathcal{E}_{max}$ to 500 and the maximum steps k_{max} for each episode to 50. The discount factor γ is set to 0.9. A batch size of 32 is chosen. From the previous example, we observed that $m_L = 8,\ m_F = 100$ performed the best and hence in this example we evaluate the performance of the proposed aSNAQ method with $m_L = 8,\ m_F = 100$ in comparison with Adam and RMSprop. For all successful solutions i.e. all nets routed within the maximum number of episodes, zero overflow was obtained and the corresponding total wirelength was calculated using the ISPD'08 contest evaluator. A summary of the results of the 30 benchmarks are shown in Table 6.1. The table shows the total wirelength (WL) if all pins were successfully routed. The *diff* column shows the amount of wirelength reduction obtained in comparison to the baseline (A* solution) wirelength. Overflow if any, are indicated within paranthesis next to the wirelength. $\mathcal{R}_{best}$ indicates the best cumulative reward obtained and $\mathcal{E}_{first}$ corresponds to the first episode when all pins are successfully routed. From the table, it can be observed that for 26 out 30 benchmarks aSNAQ was successful in routing all the pins within 500 episodes while Adam and RMSprop were successful in only 23 out of 30 and 20 out of 30 benchmarks, respectively. Furthermore, in most of the cases the routing solution obtained by aSNAQ had significant wirelength reduction compared to the baseline. Also, aSNAQ could find a routing solution for all 50 pins in fewer episodes compared to Adam and RMSprop in most cases. Figure 6.6 shows the average of all 30 benchmarks cumulative reward over 500 episodes. It can be noted that aSNAQ is faster in attaining higher average cumulative reward compared to Adam and RMSprop, thus confirming that aSNAQ is effective in training the deep Q-Network and obtaining faster convergence.

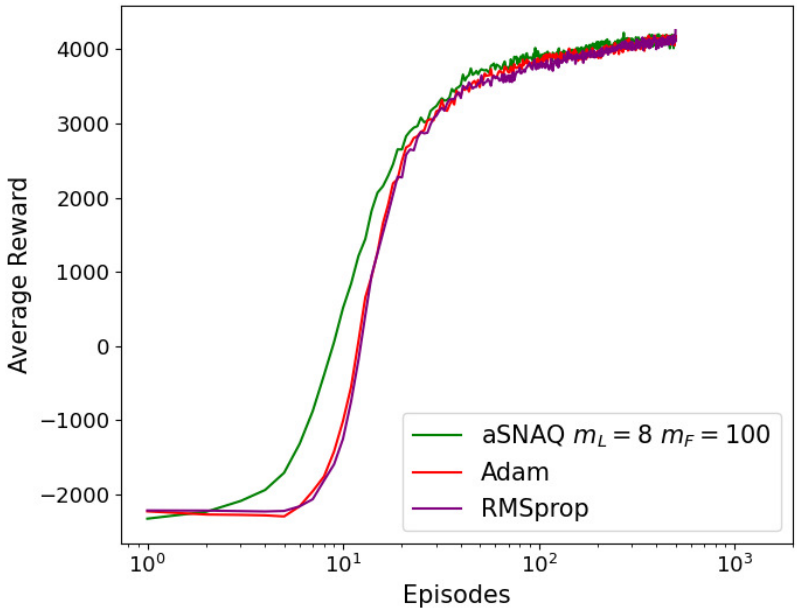


Figure 6.6: Average reward over 30 benchmarks with 50 two-pin nets.

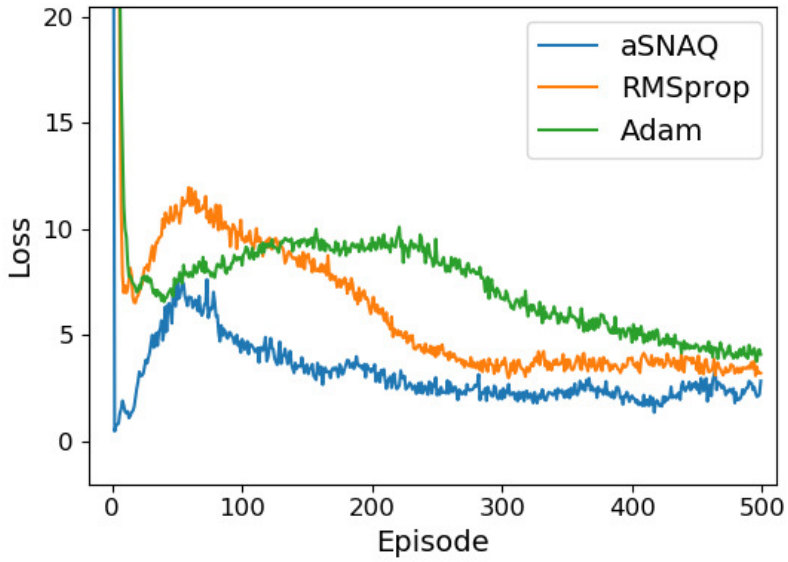


Figure 6.7: Variation of loss over episodes.

Table 6.1: Summary of the results on 30 benchmarks with 50 nets.

Netlist	A*	Adam				RMSprop				aSNAQ			
Num	WL	WL	diff	$\mathcal{R}_{best}$	$\mathcal{E}_{first}$	WL	diff	$\mathcal{R}_{best}$	$\mathcal{E}_{first}$	WL	diff	$\mathcal{R}_{best}$	$\mathcal{E}_{first}$
1	390	-	-	4386	-	-	-	4363	-	**368**	-22	4667	224
2	386	-	-	4505	-	-	-	4513	-	**376**	-10	4610	148
3	379	-	-	4234	-	-	-	4533	-	-	-	4382	-
4	369	348	-21	4690	228	350	-19	4685	151	**345**	-24	4699	75
5	366	362	-4	4679	100	**361**	-5	4681	135	369	+3	4656	458
6	352	348	-4	4691	337	344	-8	4697	222	**335**	-17	4701	106
7	430	-	-	4053	-	-	-	4322	-	-	-	4324	-
8	398	-	-	4522	-	-	-	4513	-	**377**	-21	4663	135
9	369	369	0	4669	225	**347**	-22	4687	153	348	-21	4693	64
10	366	359	-7	4674	106	375	+9	4660	223	**357**	-9	4683	67
11	379	380	+1	4660	174	380	+1	4658	262	-	-	4523	-
12	351	**346**	-5	4692	133	351	0	4689	95	348	-3	4692	58
13	395	411	+16	4616	456	397	+2	4645	180	**394**	-1	4640	87
14	340	343	+3	4700	57	**338**	-2	4706	77	341	+1	4699	49
15	375	374	-1	4659	267	384	+9	4660	211	**371**	-4	4668	195
16	386	355	-31	4682	279	-	-	4542	-	**348**	-38	4690	224
17	360	363	+3	4681	83	365	+5	4674	73	**360**	0	4680	56
18	343	334	-9	4705	81	333	-10	4712	84	**331**	-12	4714	46
19	389	385	-4	4655	164	400	+11	4615	412	**377**	-12	4657	239
20	337 (1)	337	0	4710	79	333	-4	4703	231	**333**	-4	4704	27
21	367	366	-1	4678	258	368	+1	4675	111	**352**	-15	4690	124
22	356 (2)	**366**	+10	4675	219	-	-	4528	-	-	-	4243	-
23	384	380	-4	4654	338	382	-2	4651	227	**374**	-10	4662	128
24	419 (2)	-	-	4303	-	-	-	4473	-	**388**	-31	4632	291
25	364 (2)	368	+4	4658	149	**362**	-2	4682	361	365	+1	4675	159
26	367 (3)	348	-19	4648	114	**342**	-25	4698	101	342	-25	4696	147
27	371 (2)	356	-15	4680	197	-	-	4530	-	**356**	-15	4682	220
28	399 (5)	-	-	4343	-	-	-	4482	-	**391**	-8	4640	282
29	366 (1)	**360**	-6	4682	104	363	-3	4677	140	362	-4	4678	96
30	387 (4)	371	-16	4669	139	380	-7	4638	390	**369**	-18	4666	106

6.5.3 Discussions on the performance

In several non-convex optimization problems, second-order methods have shown to speed up convergence and improve performance by effectively minimizing the loss or error function. The Q-learning method, a form of temporal difference learning, uses the Q-value function that satisfies the Bellman equation to maximize the cumulative reward. For problems with larger state-action space, DQN and double DQN methods are used to estimate the Q-values. The loss function used in the training of these DQNs is the mean-squared Bellman error as shown in (6.1), which is a non-convex function [91]. Thus, it can be speculated that a second-order method such as the aSNAQ algorithm could be effective in minimizing the mean-squared Bellman error within fewer episodes. Furthermore, in reinforcement learning using DQN, effective reduction of the error function results in better estimation of the Q-value and hence better action-selection and higher cumulative reward. It is also noteworthy that in reinforcement learning, since the training set is dynamically populated based on the state-action-reward transition, a better action-selection leads to better training samples getting accumulated in the experience replay buffer, which in turn results in efficient training of the DQN and better estimation of the Q-value. From the simulation results of the benchmarks with 10 and 50 nets, we can thus say that aSNAQ was efficient in minimizing the loss function faster, which led to faster increase in the cumulative reward. However, from Figure 5 it can be seen that as the number of episodes increase, the average cumulative reward of aSNAQ, Adam and RMSprop gradually merge. It can be speculated that on further continuing the training for several more episodes, the reward and hence the total wirelength of the routing solution obtained from DQNs trained by aSNAQ, Adam and RMSprop could all be the same. This was evident in the simulations on the benchmarks with 10 nets when trained till 500 episodes. The wirelength of the routing solution obtained from DQNs trained by aSNAQ, Adam and RMSprop were all the same. However, due to time constraints, a similar study on the benchmarks with 50 nets trained for more number of episodes could not be conducted. Nevertheless, the focus of this chapter is to emphasize on the merit of aSNAQ, that it is able to attain high cumulative rewards and hence a better routing solution within fewer episodes. This implies that the aSNAQ algorithm can reach a good solution much faster than its first-order counterparts. This can be regarded as a result of better approximations obtained through incorporating second-order information and the control heuristics.

6.6 Summary

First order gradient based methods are popular in training deep neural networks. Incorporating second-order curvature information such as the QN and NAQ methods have shown to be efficient in several supervised models. This study verified the feasibility and efficiency of our proposed adaptive Stochastic Nesterov's Accelerated Quasi-Newton (aSNAQ) method in deep reinforcement learning applications as well. The proposed algorithm was evaluated on a deep reinforcement learning (DRL) framework for solving global routing. To further enhance the performance, double DQN was implemented. The results were evaluated using the ISPD'08 constest evaluator with A* search solution as the baseline for comaprison. Also, it must be noted that the A* solution is not the global optimum since the nets are routed sequentially. This study attempted to show that the DRL framework for global routing can result in better solutions compared to A* and more particularly, the solutions can be obtained within fewer episodes when the DQN is trained using aSNAQ - a second-order algorithm as when compared

to popular first-order methods such as Adam and RMSProp. It was observed that the DRL framework with double DQNs was efficient for global routing as it resulted in considerable wirelength reduction compared to the A* search solution. The average cumulative reward plots confirmed that the aSNAQ method can accelerate the training of DQNs compared to Adam and RMSProp, and can thus be used in reinforcement learning type of applications as well. In future works, further analysis of the proposed algorithm on larger netlists, nets with multipins and study on application to other problems can be studied.

7

Accelerating Symmetric Rank-1 Quasi-Newton Method

From the chapters thus far it is evident that second-order methods show faster convergence compared to first-order methods even without acceleration techniques. And until now, we have seen the acceleration of the BFGS quasi-Newton method with Nesterov's gradient. In this chapter we further investigate the feasbility of the Nesterov's acceleration on other quasi-Newton update methods. In particular, we focus on studying the Symmetric Rank-1 (SR1) quasi-Newton method and its acceleration using the Nesterov's gradient in both the deterministic and stochastic cases. This chapter is based on results published in [92].

7.1 Introduction

Second-order methods have shown to have better convergence than first-order methods, with the only drawbacks being high computational and storage costs. Thus, several approximations have been proposed under Newton [40, 41] and quasi-Newton [42] methods to efficiently use the second-order information while keeping the computational load minimal. In neural network training, the Broyden-Fletcher-Goldfarb-Shanon (BFGS) method is the most widely studied quasi-Newton method. However, going through the evolution of quasi-Newton methods, we have other quasi-Newton update formulae prior to the BFGS update, that have not been sufficiently explored in training neural networks. The Symmetric Rank-1 (SR1) quasi-Newton method is one among them and though less commonly used in training neural networks due to its insufficient performance, is known to have interesting properties and provide good Hessian approximations when used with a trust-region approach [93, 94]. Several works in optimization [95–97] have shown SR1 quasi-Newton methods to be efficient. Recent works such as [98, 99] have proposed sampled LSR1 (limited memory) quasi-Newton updates for machine learning and describe efficient ways for distributed training implementation. Similar to the acceleration of the BFGS quasi-Newton method with Nesterov's gradient, we explore the feasibility of improving the performance of the LSR1 quasi-Newton method using Nesterov's gradient. We thus propose a novel limited memory Nesterov's accelerated symmetric rank-1 method (L-SR1-N) for training neural networks. We show that the performance of the LSR1 quasi-Newton method can be significantly improved using the trust-region approach and Nesterov's acceleration, in both the deterministic and stochastic cases.

7.2 Background

Given a subset of the training dataset $X \subseteq T_r$ with input-output pair samples $(x_p, o_p)_{p \in X}$ drawn at random from the training set T_r and error function $E_p(\mathbf{w}; x_p, o_p)$ parameterized by a vector $\mathbf{w} \in \mathbb{R}^d$, the objective function to be minimized is defined as

$$E(\mathbf{w}_k) = \frac{1}{b} \sum_{p \in X_k} E_p(\mathbf{w}_k), \tag{7.1}$$

where $X_k \subset T_r$ is the minibatch sample set of size b. In full batch, $X_k = T_r$ and $b = n$ where $n = |T_r|$. In gradient based methods, the objective function $E(\mathbf{w})$ is minimized by the iterative formula

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_{k+1}, \quad \text{for} \quad k = 1, 2, ..., k_{max} \in \mathbb{N}. \tag{7.2}$$

where $k \in \mathbb{N}$ is the iteration count and $\mathbf{v}_{k+1}$ is the update vector, which is defined for each gradient algorithm. For instance, in the gradient descent (GD) method (see Algorithm 2.1) the update vector $\mathbf{v}_{k+1}$ is given as

$$\mathbf{v}_{k+1} = -\alpha_k \nabla E(\mathbf{w}_k). \tag{7.3}$$

where α_k is the learning rate that determines the step size along the direction of the gradient $\nabla E(\mathbf{w}_k)$, and is usually fixed or set to a simple decay schedule.
The gradient descent method is simple but relatively slow in convegence and hence several accelerated methods have be proposed in the past. The Nesterov's Accelerated Gradient (NAG) method [20] is a modification of the gradient descent method in which the gradient is computed at $\mathbf{w}_k + \mu_k \mathbf{v}_k$ instead of $\mathbf{w}_k$ (see Algorithm 2.3). Thus, the update vector is given by:

$$\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k - \alpha_k \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k), \tag{7.4}$$

where $\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$ is the gradient at $\mathbf{w}_k + \mu_k \mathbf{v}_k$ and is referred to as Nesterov's accelerated gradient. The momentum coefficient μ_k is a hyperparameter chosen in the range (0,1). Several adaptive momentum and restart schemes have also been proposed for the choice of the momentum [19, 100].

7.2.1 Second-Order Quasi-Newton Methods

While first-order methods use only the gradient or first-order derivatives in updating the iterates, second-order methods such as the Newton's method use both the first and second-order derivatives, thereby ensuring better convergence than first-order methods. The update vector of second-order methods take the form

$$\mathbf{v}_{k+1} = -\alpha_k \mathbf{H}_k \nabla E(\mathbf{w}_k). \tag{7.5}$$

However, computing the inverse of the Hessian matrix $\mathbf{H}_k = \mathbf{B}_k^{-1}$ incurs a high computational cost, especially for large-scale problems. For example, the computation cost of the Newton's method is of the order $O(n^3)$. Thus, quasi-Newton methods are widely used where the inverse of the Hessian matrix is approximated iteratively using only the gradient of the objective function, which thus reduces the computational cost to the order of $O(n^2)$.

BFGS quasi-Newton Method

The Broyden-Fletcher-Goldfarb-Shanon (BFGS) algorithm (see Algorithm 3.2) is one of the most popular quasi-Newton methods for unconstrained optimization. The update vector of the BFGS quasi-Newton method is given as $\mathbf{v}_{k+1} = \alpha_k \mathbf{g}_k$, where $\mathbf{g}_k = -\mathbf{H}_k^{\text{BFGS}} \nabla E(\mathbf{w}_k)$ is the search direction. The hessian matrix $\mathbf{H}_k^{\text{BFGS}}$ is symmetric positive definite and is iteratively approximated by the following BFGS rank-2 update formula [1].

$$\mathbf{H}_{k+1}^{\text{BFGS}} = \left(\mathbf{I} - \frac{\mathbf{p}_k \mathbf{q}_k^{\text{T}}}{\mathbf{q}_k^{\text{T}} \mathbf{p}_k}\right) \mathbf{H}_k^{\text{BFGS}} \left(\mathbf{I} - \frac{\mathbf{q}_k \mathbf{p}_k^{\text{T}}}{\mathbf{q}_k^{\text{T}} \mathbf{p}_k}\right) + \frac{\mathbf{p}_k \mathbf{p}_k^{\text{T}}}{\mathbf{q}_k^{\text{T}} \mathbf{p}_k}, \tag{7.6}$$

where $\mathbf{I}$ denotes the identity matrix, and

$$\mathbf{p}_k = \mathbf{w}_{k+1} - \mathbf{w}_k \;\text{ and }\; \mathbf{q}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k). \tag{7.7}$$

Nesterov's Accelerated Quasi-Newton Method

The Nesterov's Accelerated Quasi-Newton (NAQ) [31] method (see Algorithm 3.3) introduces Nesterov's acceleration to the BFGS quasi-Newton method by approximating the quadratic model of the objective function at $\mathbf{w}_k + \mu_k \mathbf{v}_k$ and by incorporating Nesterov's accelerated gradient $\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$ in its Hessian update. The update vector of NAQ can be written as:

$$\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k + \alpha_k \mathbf{g}_k, \tag{7.8}$$

where $\mathbf{g}_k = -\mathbf{H}_k^{\text{NAQ}} \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$ is the search direction and the Hessian update equation is given as

$$\mathbf{H}_{k+1}^{\text{NAQ}} = \left(\mathbf{I} - \frac{\mathbf{p}_k \mathbf{q}_k^{\text{T}}}{\mathbf{q}_k^{\text{T}} \mathbf{p}_k}\right) \mathbf{H}_k^{\text{NAQ}} \left(\mathbf{I} - \frac{\mathbf{q}_k \mathbf{p}_k^{\text{T}}}{\mathbf{q}_k^{\text{T}} \mathbf{p}_k}\right) + \frac{\mathbf{p}_k \mathbf{p}_k^{\text{T}}}{\mathbf{q}_k^{\text{T}} \mathbf{p}_k}, \tag{7.9}$$

where

$$\mathbf{p}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k) \;\text{ and }\; \mathbf{q}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k). \tag{7.10}$$

It is shown in [31] that NAQ has similar convergence properties to that of BFGS.

SR1 Quasi-Newton Method

While the BFGS and NAQ methods update the Hessian using rank-2 updates, the Symmetric Rank-1 (SR1) method performs rank-1 updates [1]. The Hessian update of the SR1 method is given as

$$\mathbf{H}_{k+1}^{\text{SR1}} = \mathbf{H}_k^{\text{SR1}} + \frac{(\mathbf{p}_k - \mathbf{H}_k^{\text{SR1}} \mathbf{q}_k)(\mathbf{p}_k - \mathbf{H}_k^{\text{SR1}} \mathbf{q}_k)^{\text{T}}}{(\mathbf{p}_k - \mathbf{H}_k^{\text{SR1}} \mathbf{q}_k)^{\text{T}} \mathbf{q}_k}, \tag{7.11}$$

where,

$$\mathbf{p}_k = \mathbf{w}_{k+1} - \mathbf{w}_k \;\text{ and }\; \mathbf{q}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k). \tag{7.12}$$

Unlike the BFGS or NAQ method, the Hessian generated by the SR1 update may not always be positive definite. Also, the denominator can vanish or become zero. Thus, SR1 methods are not popularly used in neural network training. However, SR1 methods are known to converge faster towards the true Hessian than the BFGS method, and have computational advantages for sparse problems [93]. Furthermore, several strategies such as skipping the update or using trust region approaches have been introduced to overcome the drawbacks of the SR1 method, resulting in them performing almost on par with, if not better than, the BFGS method.

7.2.2 Trust region approach

Line search methods and trust-region methods are two fundamental strategies used to move the current point to a new iterate. Both generate steps with the help of a quadratic model of the objective function, but they use the model in different ways. Line search methods use it to generate a search direction, and then focus their efforts on finding a suitable step length α_k along this direction. Trust-region methods define a region around the current iterate within which they trust the model to be an adequate representation of the objective function, and then choose the step to be the approximate minimizer of the model in this region.

One of the key ingredients in a trust-region algorithm is the strategy for choosing the trust-region radius Δ_k at each iteration. Given a step $\mathbf{s}_k$ we define the ratio

$$\rho_k = \frac{E(\mathbf{w}_k) - E(\mathbf{w}_k + \mathbf{s}_k)}{m_k(0) - m_k(\mathbf{s}_k)} \tag{7.13}$$

where the numerator gives the actual reduction and the denominator is the reduction in error function E predicted by the model function. The predicted reduction is always nonnegative since $\mathbf{s}_k$ is obtained by minimizing the model m_k over a region that includes $\mathbf{s} = 0$. This implies that if the ratio ρ_k is negative, the objective function at the new iterate is greater than the current value and hence the step must be rejected. If ρ_k is close to 1, there is good agreement between the model m_k and the objective function over this step, so it is safe to expand the trust region for the next iteration. If ρ_k is positive but significantly smaller than 1, we do not alter the trust region, but if it is close to zero or negative, trust region radius is reduced for the next iteration. The algorithm to adjust the trust region radius is shown in Algorithm 7.1.

Algorithm 7.1 adjustTR

Require: ρ_k and trust region radius Δ_k
if $\rho_k > 0.75$ **then**
  if $\|\mathbf{s}_k\| \leq 0.8\Delta_k$ **then**
    $\Delta_{k+1} = \Delta_k$
  else
    $\Delta_{k+1} = 2\Delta_k$
  end if
else
  if $0.1 \leq \rho_k \leq 0.75$ **then**
    $\Delta_{k+1} = \Delta_k$
  else
    $\Delta_{k+1} = 0.5\Delta_k$
  end if
end if

Often, as the scale of the neural network model increases, the $O(d^2)$ cost of storing and updating the Hessian matrices $\mathbf{H}_k^{\text{SR1}}$, $\mathbf{H}_k^{\text{BFGS}}$ and $\mathbf{H}_k^{\text{NAQ}}$ become expensive. Hence, limited memory variants LSR1, LBFGS and LNAQ were proposed, and the respective Hessian matrices were updated using only the last m_L curvature information pairs $\{\mathbf{p}_i, \mathbf{q}_i\}_{i=k-1}^{k-m_L-1}$, where m_L is the limited memory size and is chosen such that $m_L \ll b$.

7.3 Proposed L-SR1-N Method

Second-order quasi-Newton (QN) methods build an approximation of a quadratic model recursively using the curvature information along a generated trajectory. In this section, we first show that the Nesterov's acceleration when applied to QN satisfies the secant condition and then show the derivation of the proposed Nesterov Accelerated Symmetric Rank-1 Quasi-Newton Method.

Nesterov Accelerated Symmetric Rank-1 Quasi-Newton Method

Suppose that $E : R^d \rightarrow R$ is continuosly differentiable and that $\mathbf{d} \in \mathbb{R}^d$, then from Taylor series, the quadratic model of the objective function at an iterate $\mathbf{w}_k$ is given as

$$E(\mathbf{w}_k + \mathbf{d}) \approx m_k(\mathbf{d}) \approx E(\mathbf{w}_k) + \nabla E(\mathbf{w}_k)^{\mathrm{T}}\mathbf{d} + \frac{1}{2}\,\mathbf{d}^{\mathrm{T}}\nabla^2 E(\mathbf{w}_k)\mathbf{d}. \tag{7.14}$$

In order to find the minimizer $\mathbf{d}_k$, we equate $\nabla \mathbf{m}_k(\mathbf{d}) = 0$ and thus have

$$\mathbf{d}_k = -\nabla^2 E(\mathbf{w}_k)^{-1}\nabla E(\mathbf{w}_k) = -\mathbf{B}_k^{-1}\nabla E(\mathbf{w}_k)\,. \tag{7.15}$$

The new iterate $\mathbf{w}_{k+1}$ is given as,

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \mathbf{B}_k^{-1}\nabla E(\mathbf{w}_k), \tag{7.16}$$

and the quadratic model at the new iterate is given as

$$E(\mathbf{w}_{k+1} + \mathbf{d}) \approx m_{k+1}(\mathbf{d}) \approx E(\mathbf{w}_{k+1}) + \nabla E(\mathbf{w}_{k+1})^{\mathrm{T}}\mathbf{d} + \frac{1}{2}\,\mathbf{d}^{\mathrm{T}}\mathbf{B}_{k+1}\mathbf{d}\,, \tag{7.17}$$

where α_k is the step length and $\mathbf{B}_k^{-1} = \mathbf{H}_k$ and its consecutive updates $\mathbf{B}_{k+1}^{-1} = \mathbf{H}_{k+1}$ are symmetric positive definite matrices satisfying the secant condition. The Nesterov's acceleration approximates the quadratic model at $\mathbf{w}_k + \mu_k\mathbf{v}_k$ instead of the iterate at $\mathbf{w}_k$. Here $\mathbf{v}_k = \mathbf{w}_k - \mathbf{w}_{k-1}$ and μ_k is the momentum coefficient in the range $(0, 1)$. Thus we have the new iterate $\mathbf{w}_{k+1}$ given as,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mu_k\mathbf{v}_k - \alpha_k \mathbf{B}_k^{-1}\nabla E(\mathbf{w}_k + \mu_k\mathbf{v}_k), \tag{7.18}$$

$$= \mathbf{w}_k + \mu_k\mathbf{v}_k + \alpha_k\mathbf{d}_k. \tag{7.19}$$

In order to show that the Nesterov accelerated updates also satisfy the secant condition, we require that the gradient of m_{k+1} should match the gradient of the objective function at the last two iterates $(\mathbf{w}_k + \mu_k\mathbf{v}_k)$ and $\mathbf{w}_{k+1}$. In other words, we impose the following two requirements on $\mathbf{B}_{k+1}$,

$$\nabla m_{k+1}|_{\mathbf{d}=0} = \nabla E(\mathbf{w}_{k+1} + \mathbf{d})|_{\mathbf{d}=0} = \nabla E(\mathbf{w}_{k+1}), \tag{7.20}$$

$$\nabla m_{k+1}|_{\mathbf{d}=-\alpha_k\mathbf{d}_k} = \nabla E(\mathbf{w}_{k+1} + \mathbf{d})|_{\mathbf{d}=-\alpha_k\mathbf{d}_k} = \nabla E(\mathbf{w}_{k+1} - \alpha_k\mathbf{d}_k) = \nabla E(\mathbf{w}_k + \mu_k\mathbf{v}_k). \tag{7.21}$$

From (7.17),

$$\nabla m_{k+1}(\mathbf{d}) = \nabla E(\mathbf{w}_{k+1}) + \mathbf{B}_{k+1}\mathbf{d}. \tag{7.22}$$

Substituting $\mathbf{d} = 0$ in (7.22), the condition in (7.20) is satisfied. From (7.21) and substituting $\mathbf{d} = -\alpha_k\mathbf{d}_k$ in (7.22), we have

$$\nabla E(\mathbf{w}_k + \mu_k\mathbf{v}_k) = \nabla E(\mathbf{w}_{k+1}) - \alpha_k\mathbf{B}_{k+1}\mathbf{d}_k. \tag{7.23}$$

Substituting for $\alpha_k \mathbf{d}_k$ from (7.19) in (7.23), we get

$$\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) = \nabla E(\mathbf{w}_{k+1}) - \mathbf{B}_{k+1}(\mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k)). \tag{7.24}$$

On rearranging the terms, we have the secant condition

$$\mathbf{y}_k = \mathbf{B}_{k+1}\mathbf{s}_k, \tag{7.25}$$

where,

$$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) \text{ and } \mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k) = \alpha_k \mathbf{d}_k. \tag{7.26}$$

We have thus shown that the Nesterov accelerated QN update satisfies the secant condition. The update equation of $\mathbf{B}_{k+1}$ for SR1-N can be derived similarly to that of the classical SR1 update [1]. The secant condition requires $\mathbf{B}_k$ to be updated with a symmetric matrix such that $\mathbf{B}_{k+1}$ is also symmetric and satisfies the secant condition. The update of $\mathbf{B}_{k+1}$ is defined using a symmetric-rank-1 matrix formed by an arbitrary vector $\mathbf{u}\mathbf{u}^{\mathrm{T}}$ is given as

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \sigma \mathbf{u}\mathbf{u}^{\mathrm{T}}, \tag{7.27}$$

where σ and $\mathbf{u}$ are chosen such that they satisfy the secant condtion in (7.25). Substituting (7.27) in (7.25), we get

$$\mathbf{y}_k = \mathbf{B}_k \mathbf{s}_k + (\sigma \mathbf{u}^{\mathrm{T}} \mathbf{s}_k)\mathbf{u}. \tag{7.28}$$

Since $(\sigma \mathbf{u}^{\mathrm{T}} \mathbf{s}_k)$ is a scalar, we can deduce $\mathbf{u}$ a scalar multiple of $\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k$ and thus have

$$(\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k) = \sigma \delta^2 [\mathbf{s}_k^{\mathrm{T}} (\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)](\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k), \tag{7.29}$$

where

$$\sigma = \mathrm{sign}[\mathbf{s}_k^{\mathrm{T}} (\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)] \quad \text{and} \quad \delta = \pm |[\mathbf{s}_k^{\mathrm{T}} (\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)]|^{1/2} . \tag{7.30}$$

Thus the proposed Nesterov accelerated symmetric rank-1(L-SR1-N) update is given as

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{(\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)(\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)^{\mathrm{T}}}{(\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)^{\mathrm{T}} \mathbf{s}_k} . \tag{7.31}$$

Note that the Hessian update is performed only if the below condition in (7.32) is satisfied, otherwise $\mathbf{B}_{k+1} = \mathbf{B}_k$.

$$|\mathbf{s}_k^{\mathrm{T}} (\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)| \geq \rho \, \|\mathbf{s}_k\| \, \|\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k\| . \tag{7.32}$$

By applying the Sherman-Morrison-Woodbury Formula [1], we can find $\mathbf{B}_{k+1}^{-1} = \mathbf{H}_{k+1}$ as

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{(\mathbf{s}_k - \mathbf{H}_k \mathbf{y}_k)(\mathbf{s}_k - \mathbf{H}_k \mathbf{y}_k)^{\mathrm{T}}}{(\mathbf{s}_k - \mathbf{H}_k \mathbf{y}_k)^{\mathrm{T}} \mathbf{y}_k}, \tag{7.33}$$

where,

$$\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) \text{ and } \mathbf{s}_k = \mathbf{w}_{k+1} - (\mathbf{w}_k + \mu_k \mathbf{v}_k) = \alpha_k \mathbf{d}_k. \tag{7.34}$$

The proposed algorithm is as shown in Algorithm 7.2. We implement the proposed method in its limited memory form, where the Hessian is updated using the recent m_L curvature information pairs satisfying (7.32). Here m_L denotes the limited memory size and is chosen such that $m_L \ll b$. The proposed method uses the trust-region approach where the subproblem is solved using the CG-Steihaug method [1] as shown in Algorithm 7.3. Also note that the proposed L-SR1-N has two gradient

computations per iteration. The Nesterov's gradient $\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_\mathbf{k})$ can be approximated [33, 101] as a linear combination of past gradients as shown below.

$$\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) \approx (1 + \mu_k)\nabla E(\mathbf{w}_k) - \mu_k \nabla E(\mathbf{w}_{k-1}). \tag{7.35}$$

Thus we have the momentum accelerated symmetric rank-1 (L-MoSR1) method by approximating the Nesterov's gradient in L-SR1-N.

Algorithm 7.2 L-SR1-N Method

while $\|\nabla E(\mathbf{w}_k)\| > \epsilon$ and $k < k_{\max}$ **do**
  Determine μ_k
  Compute $\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$
  Find $\mathbf{s}_k$ by CG-Steihaug subproblem solver in Algorithm (7.3)
  Compute $\rho_k = \frac{E(\mathbf{w}_k + \mu_k \mathbf{v}_k) - E(\mathbf{w}_k + \mu_k \mathbf{v}_k + \mathbf{s}_k)}{m_k(0) - m_k(\mathbf{s}_k)}$
  if $\rho_k \geq \eta$ **then**
    Set $\mathbf{v}_{k+1} = \mu_k \mathbf{v}_k + \mathbf{s}_k, \mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{v}_{k+1}$
  else
    Set $\mathbf{v}_{k+1} = \mathbf{v}_k, \quad \mathbf{w}_{k+1} = \mathbf{w}_k$, reset μ_k
  end if
  $\Delta_{k+1} = \text{adjustTR}(\Delta_k, \rho_k)$
  Compute $\mathbf{y}_k = \nabla E(\mathbf{w}_{k+1}) - \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k) + \zeta \mathbf{s}_k$
  Update $(\mathbf{S}_k, \mathbf{Y}_k)$ buffer with $(\mathbf{s}_k, \mathbf{y}_k)$ if (7.32) is satisfied
end while

Algorithm 7.3 CG-Steihaug

Require: Gradient $\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$, tolerance $\epsilon_k > 0$, and trust-region radius Δ_k.
Initialize: Set $\mathbf{z}_0 = 0, \quad \mathbf{r}_0 = \nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k), \quad \mathbf{d}_0 = -\mathbf{r}_0 = -\nabla E(\mathbf{w}_k + \mu_k \mathbf{v}_k)$

if then $\|\mathbf{r}_0\| < \epsilon_k$
  return $\mathbf{s}_k = \mathbf{z}_0 = 0$
end if
for $i = 0, 1, 2, \ldots$ **do**
  if $\mathbf{d}_i^{\mathrm{T}} \mathbf{B}_k \mathbf{d}_i \leq 0$ **then**
    Find τ such that $\mathbf{s}_k = \mathbf{z}_i + \tau \mathbf{d}_i$ minimizes (7.42) and satisfies $\|\mathbf{s}_k\| = \Delta_k$
    return $\mathbf{s}_k$
  end if
  Set $\alpha_i = \frac{\mathbf{r}_i^{\mathrm{T}} \mathbf{r}_i}{\mathbf{d}_i^{\mathrm{T}} \mathbf{B}_k \mathbf{d}_i}$
  Set $\mathbf{z}_{i+1} = \mathbf{z}_i + \alpha_i \mathbf{d}_i$
  if $\|\mathbf{z}_{i+1}\| \geq \Delta_k$ **then**
    Find $\tau \geq 0$ such that $\mathbf{s}_k = \mathbf{z}_i + \tau \mathbf{d}_i$ satisfies $\|\mathbf{s}_k\| = \Delta_k$
    return $\mathbf{s}_k$
  end if
  Set $\mathbf{r}_{i+1} = \mathbf{r}_i + \alpha_i \mathbf{B}_k \mathbf{d}_i$
  if $\|\mathbf{r}_{i+1}\| < \epsilon_k$ **then**
    return $\mathbf{s}_k = \mathbf{z}_{i+1}$
  end if
  Set $\beta_{i+1} = \frac{\mathbf{r}_i^{\mathrm{T}} \mathbf{r}_{i+1}}{\mathbf{r}_i^{\mathrm{T}} \mathbf{r}_i}$
  Set $\mathbf{d}_{i+1} = -\mathbf{r}_{i+1} + \beta_{i+1} \mathbf{d}_i$
end for

7.4 Convergence Analysis

In this section we discuss the convergence proof of the proposed Nesterov accelerated Symmetric Rank-1 (L-SR1-N) algorithm in its limited memory form. As mentioned earlier, the Nesterov's acceleration approximates the quadratic model at $\mathbf{w}_k + \mu_k \mathbf{v}_k$ instead of the iterate at $\mathbf{w}_k$. For ease of representation, we write $\mathbf{w}_k + \mu_k \mathbf{v}_k = \hat{\mathbf{w}}_k \; \forall \; k = 1, 2, ..., k_{max} \in \mathbb{N}$. In the limited memory scheme, the Hessian matrix can be implicitly constructed using the recent m_L number of curvature information pairs $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-1}^{k-m_L-1}$. At a given iteration k, we define matrices $\mathbf{S}_k$ and $\mathbf{Y}_k$ of dimensions $d \times m_L$ as

$$\mathbf{S}_k = [\mathbf{s}_{k-1}, \mathbf{s}_{k-2}, ..., \mathbf{s}_{k-m_L-1}] \quad \text{and} \quad \mathbf{Y}_k = [\mathbf{y}_{k-1}, \mathbf{y}_{k-2}, ..., \mathbf{y}_{k-m_L-1}], \tag{7.36}$$

where the curvature pairs $\{\mathbf{s}_i, \mathbf{y}_i\}_{i=k-1}^{k-m_L-1}$ are each vectors of dimensions $d \times 1$. The Hessian approximation in (7.31) can be expressed in its compact representation form [102] as

$$\mathbf{B}_k = \mathbf{B}_0 + (\mathbf{Y}_k - \mathbf{B}_0 \mathbf{S}_k)(\mathbf{L}_k + \mathbf{D}_k + \mathbf{L}_k^{\mathrm{T}} - \mathbf{S}_k^{\mathrm{T}} \mathbf{B}_0 \mathbf{S}_k)^{-1} (\mathbf{Y}_k - \mathbf{B}_0 \mathbf{S}_k), \tag{7.37}$$

where $\mathbf{B}_0$ is the initial $d \times d$ Hessian matrix, $\mathbf{L}_k$ is a $m_L \times m_L$ lower triangular matrix and $\mathbf{D}_k$ is a $m_L \times m_L$ diagonal matrix as given below,

$$\mathbf{B}_0 = \gamma_k \mathbf{I},$$

$$(\mathbf{L}_k)_{i,j} = \begin{cases} \mathbf{s}_i^{\mathrm{T}} \mathbf{y}_j & \text{if } i > j, \\ 0 & \text{otherwise,} \end{cases}$$

$$\mathbf{D}_k = \text{diag } [\mathbf{S}_k^{\mathrm{T}} \mathbf{Y}_k]. \tag{7.38}$$

Let $\mathbf{\Omega}$ be the level set such that $\mathbf{\Omega} = \{\mathbf{w} \in \mathbb{R}^d \; : \; E(\mathbf{w}) \leq E(\mathbf{w_0})\}$ and $\{\mathbf{s}_k\} \; \forall \; k = 1, 2, ..., k_{max} \in \mathbb{N}$, denote the sequence generated by the explicit trust-region algorithm where Δ_k be the trust-region radius of the successful update step. We choose $\gamma_k = 0$. Since the curvature information pairs $(\mathbf{s}_k, \mathbf{y}_k)$ given by (7.34) are stored in $\mathbf{S}_k$ and $\mathbf{Y}_k$ only if they satisfy the condition in (7.32), the matrix $\mathbf{M}_k = (\mathbf{L}_k + \mathbf{D}_k + \mathbf{L}_k^{\mathrm{T}} - \mathbf{S}_k^{\mathrm{T}} \mathbf{B}_0 \mathbf{S}_k)$ is invertible and positive semi-definite.

Assumption 7.4.1. *The sequence of iterates $\mathbf{w}_k$ and $\hat{\mathbf{w}}_k \; \forall \; k = 1, 2, ..., k_{max} \in \mathbb{N}$ remains in the closed and bounded set $\mathbf{\Omega}$ on which the objective function is twice continuously differentiable and has Lipschitz continuous gradient, i.e., there exists a constant $L > 0$ such that*

$$\|\nabla E(\mathbf{w}_{k+1}) - \nabla E(\hat{\mathbf{w}}_k)\| \leq L \|\mathbf{w}_{k+1} - \hat{\mathbf{w}}_k\| \quad \forall \; \mathbf{w}_{k+1}, \; \hat{\mathbf{w}}_k \in \mathbb{R}^d. \tag{7.39}$$

Assumption 7.4.2. *The Hessian matrix is bounded and well-defined, .i.e, there exists constants ρ and M, such that*

$$\rho \leq \|\mathbf{B}_k\| \leq M \quad \forall \; k = 1, 2, ..., k_{max} \in \mathbb{N}. \tag{7.40}$$

and for each iteration k

$$|\mathbf{s}_k^{\mathrm{T}} (\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k)| \geq \rho \; \|\mathbf{s}_k\| \; \|\mathbf{y}_k - \mathbf{B}_k \mathbf{s}_k\|. \tag{7.41}$$

Assumption 7.4.3. *Let $\mathbf{B}_k$ be any $n \times n$ symmetric matrix and $\mathbf{s}_k$ be an optimal solution to the trust region subproblem,*

$$\min_{\mathbf{d}} \; m_k(\mathbf{d}) = E(\hat{\mathbf{w}}_k) + \mathbf{d}^{\mathrm{T}} \nabla E(\hat{\mathbf{w}}_k) + \frac{1}{2}\mathbf{d}^{\mathrm{T}} \mathbf{B}_k \mathbf{d}, \tag{7.42}$$

where $\hat{\mathbf{w}}_k + \mathbf{d}$ lies in the trust region. Then for all $k \geq 0$,

$$\left| \nabla E(\hat{\mathbf{w}}_k)^{\mathrm{T}} \mathbf{s}_k + \frac{1}{2}\mathbf{s}_k^{\mathrm{T}} \mathbf{B}_k \mathbf{s}_k \right| \geq \frac{1}{2} \left|\left| \nabla E(\hat{\mathbf{w}}_k) \right|\right| \min \left\{ \Delta_k, \frac{\| \nabla E(\hat{\mathbf{w}}_k) \|}{\| \mathbf{B}_k \|} \right\}. \tag{7.43}$$

This assumption ensures that the subproblem solved by trust-region results in a sufficiently optimal solution at every iteration. The proof for this assumption can be shown similar to the trust-region proof by Powell.

Lemma 7.4.1. *If Assumptions 7.4.1 to 7.4.3 hold, and $\mathbf{s}_k$ be an optimal solution to the trust region subproblem given in (7.42), and if the initial γ_k is bounded (i.e., $0 \leq \gamma_k \leq \bar{\gamma}_k$), then for all $k \geq 0$, the Hessian update given by Algorithm 7.2 and (7.27) is bounded.*

Proof. We begin with the proof for the general case [103], where the Hessian is bounded by

$$\| \mathbf{B}_k^{(j)} \| \leq \left(1 + \frac{1}{\rho}\right)^j \gamma_k + \left[\left(1 + \frac{1}{\rho}\right)^j - 1 \right] M. \tag{7.44}$$

The proof for (7.44) is given by mathematical induction. Let m_L be the limited memory size and $(\mathbf{s}_{k,j}, \mathbf{y}_{k,j})$ be the curvature information pairs given by (7.34) at the k^{th} iteration for $j = 1, 2, ..., m_L$. For $j = 0$, we can see that (7.44) holds true. Let us assume that (7.44) holds true for some $j > 0$. Thus for $j + 1$ we have

$$\mathbf{B}_k^{(j+1)} = \mathbf{B}_k^{(j)} + \frac{\left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)\left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)^{\mathrm{T}}}{\left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)^{\mathrm{T}} \mathbf{s}_{k,j+1}} \tag{7.45}$$

$$\left|\left| \mathbf{B}_k^{(j+1)} \right|\right| \leq \left|\left| \mathbf{B}_k^{(j)} \right|\right| + \left|\left| \frac{\left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)\left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)^{\mathrm{T}}}{\left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)^{\mathrm{T}} \mathbf{s}_{k,j+1}} \right|\right| \tag{7.46}$$

$$\leq \left|\left| \mathbf{B}_k^{(j)} \right|\right| + \frac{\left|\left| \left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)\left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right)^{\mathrm{T}} \right|\right|}{\rho \left|\left| \left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right) \right|\right| \left|\left| \mathbf{s}_{k,j+1} \right|\right|} \tag{7.47}$$

$$\leq \left|\left| \mathbf{B}_k^{(j)} \right|\right| + \frac{\left|\left| \left(\mathbf{y}_{k,j+1} - \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1}\right) \right|\right|}{\rho \left|\left| \mathbf{s}_{k,j+1} \right|\right|} \tag{7.48}$$

$$\leq \left|\left| \mathbf{B}_k^{(j)} \right|\right| + \frac{\left|\left| \mathbf{y}_{k,j+1} \right|\right|}{\rho \left|\left| \mathbf{s}_{k,j+1} \right|\right|} + \frac{\left|\left| \mathbf{B}_k^{(j)} \mathbf{s}_{k,j+1} \right|\right|}{\rho \left|\left| \mathbf{s}_{k,j+1} \right|\right|} \tag{7.49}$$

$$\leq \left|\left| \mathbf{B}_k^{(j)} \right|\right| + \frac{\left|\left| \mathbf{y}_{k,j+1} \right|\right|}{\rho \left|\left| \mathbf{s}_{k,j+1} \right|\right|} + \frac{\left|\left| \mathbf{B}_k^{(j)} \right|\right|}{\rho} \tag{7.50}$$

$$\leq \left(1 + \frac{1}{\rho}\right) \left|\left| \mathbf{B}_k^{(j)} \right|\right| + \frac{M}{\rho} \tag{7.51}$$

$$\leq \left(1+\frac{1}{\rho}\right)\left[\left(1+\frac{1}{\rho}\right)^{j}\gamma_k + \left[\left(1+\frac{1}{\rho}\right)^{j}-1\right]M\right] + \frac{M}{\rho} \tag{7.52}$$

$$\left\|\mathbf{B}_k^{(j+1)}\right\| \leq \left(1+\frac{1}{\rho}\right)^{j+1}\gamma_k + \left[\left(1+\frac{1}{\rho}\right)^{j+1}-1\right]M \tag{7.53}$$

Since we use the limited memory scheme, $\mathbf{B}_{k+1} = \mathbf{B}_k^{(m_L)}$, where m_L is the limited memory size. Therefore, the Hessian approximation at the k^{th} iteration satisfies

$$\left\|\mathbf{B}_{k+1}\right\| \leq \left(1+\frac{1}{\rho}\right)^{m_L}\gamma_k + \left[\left(1+\frac{1}{\rho}\right)^{m_L}-1\right]M \tag{7.54}$$

We choose $\gamma_k = 0$ as it removes the choice of the hyperparameter for the initial Hessian $\mathbf{B}_k^{(0)} = \gamma_k \mathbf{I}$ and also ensures that the subproblem solver CG algorithm (Algorithm 7.3) terminates in at most m_L iterations [98]. Thus the Hessian approximation at the k^{th} iteration satisfies (7.55) and is still bounded.

$$\left\|\mathbf{B}_{k+1}\right\| \leq \left[\left(1+\frac{1}{\rho}\right)^{m_L}-1\right]M \tag{7.55}$$

This completes the inductive proof. □

Theorem 7.4.1. *Given a level set* $\mathbf{\Omega} = \{\mathbf{w} \in \mathbb{R}^d : E(\mathbf{w}) \leq E(\mathbf{w_0})\}$ *that is bounded, let* $\{\mathbf{w}_k\}$ *be the sequence of iterates generated by Algorithm 7.2. If Assumptions 7.4.1 to 7.4.3 holds true, then we have,*

$$\lim_{k\to\infty} \|\nabla E(\mathbf{w}_k)\| = 0. \tag{7.56}$$

Proof. From the derivation of the proposed L-SR1-N algorithm, it is shown that the Nesterov's acceleration to quasi-Newton method satisfies the secant condition. The proposed algorithm ensures the definiteness of the Hessian update as the curvature pairs used in the Hessian update satisfies (7.32) for all k. The sequence of updates are generated by solving using the trust region method where $\mathbf{s}_k$ is the optimal solution to the subproblem in (7.42). From Theorem 2.2 in [104], it can be shown that the updates made by the trust region method converges to a stationary point. Since $\mathbf{B}_k$ is shown to be bounded (Lemma 7.4.1), it follows from that theorem that as $k \to \infty$, $\mathbf{w}_k$ converges to a point such that $\|\nabla E(\mathbf{w}_k)\| = 0$. □

7.5 Simulation Results

We evaluate the performance of the proposed Nesterov accelerated symmetric rank-1 quasi-Newton (L-SR1-N) method in its limited memory form in comparison to conventional first-order methods and second-order methods. We illustrate the performances in both full batch and stochastic/mini-batch setting. The hyperparameters are set to their default values. The momentum coefficient μ_k is set to 0.9 in NAG and 0.85 in oLNAQ [37]. For L-NAQ [32], L-MoQ [28], and the proposed methods, the momentum coefficient μ_k is set adaptively. The adaptive μ_k is obtained from the following equations, where $\theta_k = 1$ and $\eta = 10^{-6}$.

$$\mu_k = \theta_k(1-\theta_k)/(\theta_k^2 + \theta_{k+1}), \tag{7.57}$$

$$\theta_{k+1}^2 = (1-\theta_{k+1})\theta_k^2 + \eta\theta_{k+1}. \tag{7.58}$$

7.5.1 Results of the Levy Function Approximation Problem

Consider the following Levy function approximation problem to be modeled by a neural network.

$$f(x_1 \dots x_p) = \frac{\pi}{p}\Big\{ \sum_{i=1}^{p-1}[(x_i - 1)^2(1 + 10\sin^2(\pi x_{i+1}))] + 10\sin^2(\pi x_1) + (x_p - 1)^2 \Big\}, x_i \in [-4, 4], \forall i. \tag{7.59}$$

We begin with evaluating the performance of the proposed L-SR1-N and L-MoSR1 in the deterministic (full batch) case, using the Levy function (7.59) example where $p = 5$. Therefore the inputs to the neural network is $\{x_1, x_2, ..., x_5\}$. We use a single hidden layer with 50 hidden neurons. The neural network architecture is thus $5 - 50 - 1$. We terminate the training at $k_{\max} = 10{,}000$, and set $\epsilon = 10^{-6}$ and $m_L = 10$. Sigmoid and linear activation functions are used for the hidden and output layers, respectively. Mean squared error function is used. The number of parameters is $d = 351$. Note that we use full batch for the training in this example and the number of training samples is $n = 5000$. Figure 7.1 shows the average results of 30 independent trials. The results confirm that the proposed L-SR1-N and L-MoSR1 have better performance compared to the first-order methods as well as the conventional LSR1 and rank-2 LBFGS quasi-Newton method. Furthermore, it can be observed that incorporating the Nesterov's gradient in LSR1 has significantly improved the performance, bringing it almost equivalent to the rank-2 Nesterov accelerated L-NAQ and momentum accelerated L-MoQ methods. Thus we can confirm that the limited memory symmetric rank-1 quasi-Newton method can be significantly accelerated using the Nesterov's gradient. From the iterations vs. training error plot, we can observe that the L-SR1-N and L-MoSR1 are almost similar in performance. This verifies that the approximation applied to L-SR1-N in L-MoSR1 is valid, and has an advantage in terms of

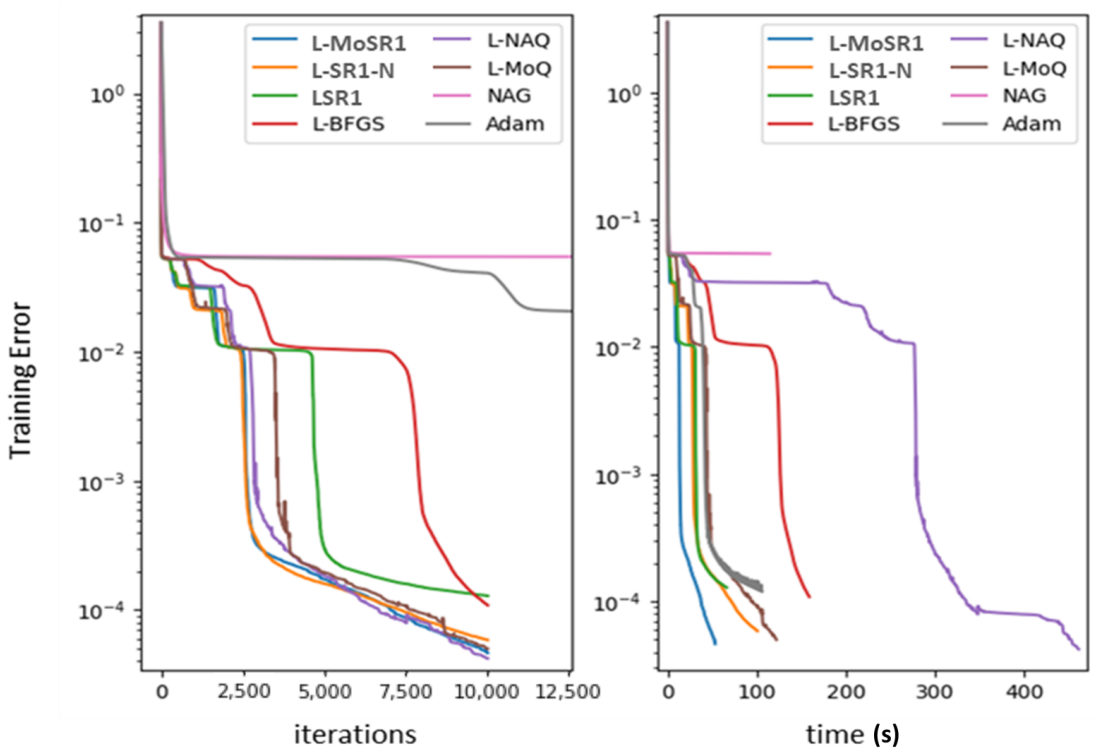


Figure 7.1: Average results on levy function approximation problem with $m_L = 10$ (full batch).

computation wall time. This can be observed in the time vs. training error plot, where the L-MoSR1 method converges much faster compared to the other first and second-order methods under comparison.

7.5.2 Results of MNIST Image Classification Problem

In large scale optimization problems, owing to the massive amount of data and large number of parameters of the neural network model, training the neural network using full batch is not feasible. Hence a stochastic approach is more desirable where the neural networks are trained using a relatively small subset of the training data, thereby significantly reducing the computational and memory requirements. However, getting second-order methods to work in a stochastic setting is a challenging task. A common problem in stochastic/mini-batch training is the sampling noise that arises due to the gradients being estimated on different mini-batch samples at each iteration. In this section, we evaluate the performance of the proposed L-SR1-N and L-MoSR1 methods in the stochastic/mini-batch setting. We use the MNIST handwritten digit image classification problem [57] for the evaluation. The MNIST dataset consists of 50,000 train and 10,000 test samples of 28×28 pixel images of handwritten digits from 0 to 9 that needs to be classified. We evaluate the performance of this image classification task on a simple fully connected neural network and LeNet-5 architectures. In a stochastic setting, the conventional LBFGS method is known to be affected by sampling noise and to alleviate this issue, [44] proposed the oLBFGS method that computes two gradients per iteration. We thus compare the performance of our proposed method against both the naive stochastic LBFGS (denoted here as oLBFGS-1) and the oLBFGS proposed in [44].

Results of MNIST on Fully Connected Neural Networks

We first consider a simple fully connected neural network with two hidden layers with 100 and 50 hidden neurons respectively. Thus, the neural network architecture used is $784 - 100 - 50 - 10$. The hidden layers use the ReLU activation function and the loss function used is the softmax cross-entropy loss function. Figure 7.2 shows the performance comparison with a batch size $b = 128$ and limited memory size of $m_L = 8$. It can be observed that the second-order quasi-Newton methods show fast convergence compared to first-order methods in the first 500 iterations. From the results we can see that even though the stochastic L-SR1-N (oL-SR1-N) and stochastic MoSR1 (oL-MoSR1) does not perform the best on the small network, it has significantly improved the performance of the stochastic LSR1 (oLSR1) method, and performs better than the oLBFGS-1 method. Since our aim is to investigate the effectiveness of the Nesterov's acceleration on SR1, we focus on the performance comparison of oLBFGS-1, oLSR1 and the proposed oL-SR1-N and oL-MoSR1 methods. As seen from Figure 7.2, oLBFGS-1, oLSR1 does not further improve the test accuracy or test loss after 1000 iterations. However, incorporating Nesterov's acceleration significantly improved the performance compared to the conventional oL-SR1 and oLBFGS-1, thus confirming the effectiveness of Nesterov's acceleration on LSR1 in the stochastic setting.

Results of MNIST on LeNet-5 Architecture

Next, we evaluate the performance of the proposed methods on a bigger network with convolutional layers. The LeNet-5 architecture consists of two sets of convolutional and average pooling layers,

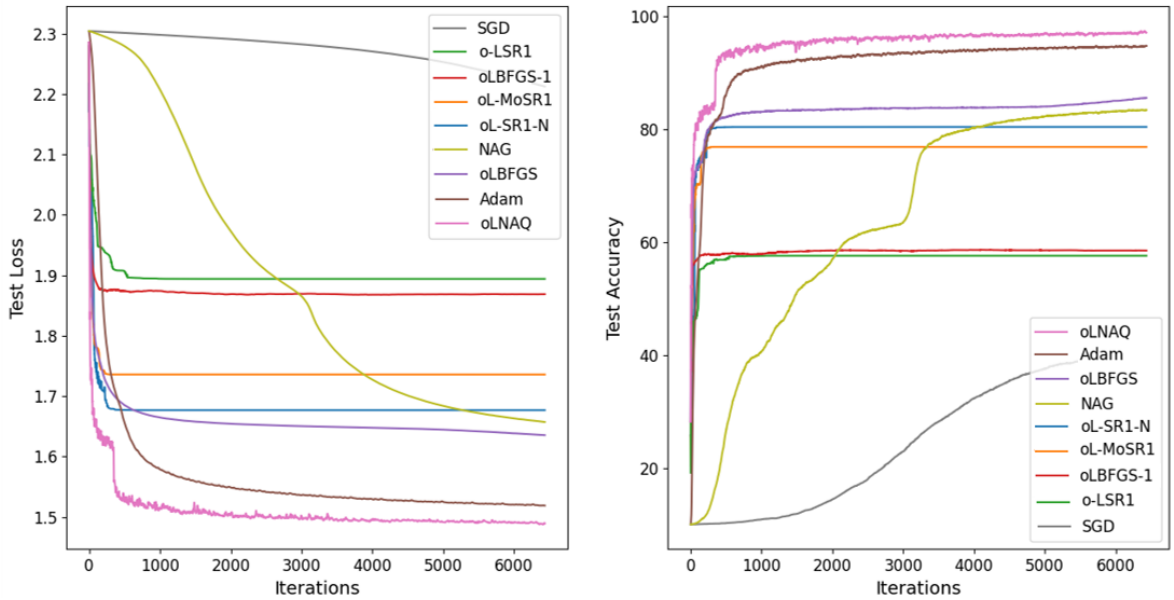


Figure 7.2: Results of MNIST on fully connected neural network with $b = 128$ and $m_L = 8$.

followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier. The number of parameters is $d = 61{,}706$. Figure 7.3 shows the performance comparison when trained with a batch size of $b = 256$ and limited memory $m_L = 8$. From the results, we can observe that oLNAQ performs the best. However, the proposed oL-SR1-N method performs better compared to both the first-order SGD, NAG, Adam and second-order oLSR1, oLBFGS-1 and oLBFGS methods. It can be confirmed that incorporating the Nesterov's gradient can accelerate and significantly improve the performance of the conventional LSR1 method, even in the stochastic setting.

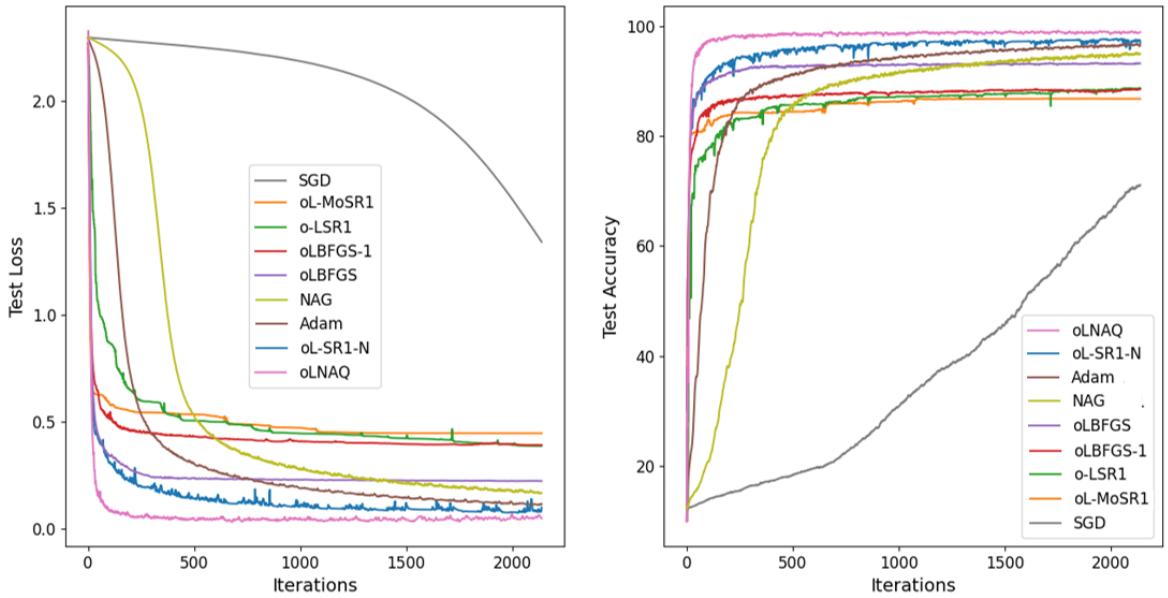


Figure 7.3: Results of MNIST on LeNet-5 architecture with $b = 256$ and $m_L = 8$.

7.6 Summary

Acceleration techniques such as the Nesterov's acceleration have shown to speed up convergence as in the cases of NAG accelerating GD and NAQ accelerating the BFGS methods. Second-order methods are said to achieve better convergence compared to first-order methods and are more suitable for parallel and distributed implementations. While the BFGS quasi-Newton method is the most extensively studied method in the context of deep learning and neural networks, there are other methods in the quasi-Newton family, such as the Symmetric Rank-1 (SR1), which are shown to be effective in optimization but not extensively studied in the context of neural networks. SR1 methods converge towards the true Hessian faster than BFGS and have computational advantages for sparse or partially separable problems [93]. Thus, investigating acceleration techniques on the SR1 method is significant. The Nesterov's acceleration is shown to accelerate convergence as seen in the case of NAQ, improving the performance of BFGS. We investigate whether the Nesterov's acceleration can improve the performance of other quasi-Newton methods such as SR1 and compare the performance among second-order Nesterov's accelerated variants. To this end, we have introduced a new limited memory Nesterov accelerated symmetric rank-1 (L-SR1-N) method for training neural networks. We compared the results with LNAQ to give a sense of comparison of how the Nesterov's acceleration affects the two methods of the quasi-Newton family, namely BFGS and SR1. The results confirm that the performance of the LSR1 method can be significantly improved in both the full batch and the stochastic settings by introducing Nesterov's accelerated gradient. Furthermore, it can be observed that the proposed L-SR1-N method is competitive with LNAQ and is substantially better than the first-order methods and second-order LSR1 and LBFGS method. It is shown both theoretically and empirically that the proposed L-SR1-N converges to a stationary point. From the results, it can also be noted that, unlike in the full batch example, the performance of oL-SR1-N and oL-MoSR1 do not correlate well in the stochastic setting. This can be regarded as due to the sampling noise, similar to that of oLBFGS-1 and oLBFGS. In the stochastic setting, the curvature information vector $\mathbf{y}_k$ of oL-MoSR1 is approximated based on the gradients computed on different mini-batch samples. This could introduce sampling noise and hence result in oL-MoSR1 not being a close approximation of the stochastic oL-SR1-N method. Future works could involve solving the sampling noise problem with multi-batch strategies such as in [105], and further improving the performance of L-SR1-N. Furthermore, a detailed study on larger networks and problems with different hyperparameter settings could test the limits of the proposed method.

8

Conclusion

8.1 Summary

This thesis presented the study of stochastic second-order quasi-Newton methods with momentum and Nesterov's acceleration for training neural networks. As a precursor, the thesis discussed the fundamentals of first and second-order methods in the context of training neural networks in detail under full batch training strategies, with performance evaluations on problems that exhibited high non-linearity and non-smoothness. In full batch training approach where at each iteration the weight updates are based on the function and gradient evaluations over the entire training set, we could confirm that that second order methods have faster convergence compared to first order methods. However the per-iteration time of second order methods is still larger than that of first order methods. Nevertheless, since first order methods require several more passes (epochs or iterations) over the dataset, second order methods still show better advantage, especially in terms of escaping local minima and saddle points. Moreover incorporating the momentum term in second order methods showed promising acceleration.

To meet with the growing demand for large-scale optimization, a family of new stochastic accelerated quasi-Newton methods have been proposed in this thesis. Getting quasi-Newton methods to work in stochastic settings has been challenging and suitable modifications and extensions were proposed and discussed. An extension of the momentum and Nesterov's accelerated BFGS quasi-Newton method in stochastic setting in full and limited memory was proposed. The proposed methods were evaluated in comparison with the oBFGS method and the results showed better performance. The proposed methods were also theoretically shown to converge at a linear rate and computation cost to be comparable to the oBFGS method.

We further extended the study of accelerated stochastic quasi-Newton methods to training recurrent neural networks. An adaptive stochastic Nesterov's Accelerated Quasi-Newton (aSNAQ) method was proposed with the focus of solving the vanishing exploding gradient issue. The proposed method used a simple adaptive step size and momentum selection schemes. Furthermore, direction normalization, weight aggregation and accumulated Fisher Information Matrix (aFIM) for weight updates and Hessian calculation was used. This method showed improved performance in training recurrent neural networks. The proposed method was shown to converge with linear rate. We further adapted the aSNAQ method to deep Q-networks. The robustness of the aSNAQ method was confirmed with an example of VLSI global routing problem using deep reinforcement learning framework.

Finally we confirmed the acceleration of the symmetric rank-1 method using Nesterov's gradient. The performance of the proposed method significantly improved the performance of the conventional symmetric rank-1 method. Also, the convergence guarantee with a trust region approach was presented for the proposed method.

8.2 Limitations and Future Work

Training of neural networks are currently dominated by first-order methods due to their simplicity and low computational complexity. However, first-order methods show slow convergence on highly non-linear problems. Second-order quasi-Newton methods exhibit fast convergence despite their high computational cost.

It must be noted that though optimization methods play an important role in training neural networks, it alone may not be sufficient to improve the overall performance of a neural network model. The overall performance of a neural network model is dependent on several factors such as the network architecture, regularization techniques such as batch normalization and dropout, optimizer, weight initialization, hyperparameters selected, etc. However, since this thesis focused on improving the performance of neural networks by accelerating the convergence rate of the optimization algorithm used in training neural networks, we showed the performance evaluations on simple datasets and small neural network structures without much fine tuning. Further evaluations on larger problems and networks could test the limits of the proposed methods.

Although this study showed that the proposed stochastic quasi-Newton methods in limited memory forms have moderate computational cost in the order of $O(d)$ which are comparable to first-order methods, as the number of parameter of the network increases, this cost can still be expensive. However, it is also notable that second order methods are more compatible for parallel and distributed implementations as there are several independent sections of the algorithms itself that can be implemented in parallel, thus reducing the per-iteration time. Thus some of the future research directions include that of extending the proposed accelerated quasi-Newton methods to parallel and distributed implementations similar to the works in [106, 107]. Also further efficient GPU programming can bring down the CPU time thus making it efficient for larger neural network models such as transformers or GPT. Efficient techniques that reduce overheads such as [108, 109] can be incorporated along with the momentum and Nesterov's acceleration.

Furthermore, though the stochastic momentum accelerated methods introduced in Chapters 4 and 7 were proposed to reduce the cost of computing two gradients per iteration, in stochastic training it may be subject to stochastic noise as the curvature information is estimated from gradients computed on different mini-batch samples. Future works could involve solving the sampling noise problem with overlapping and multi-batch strategies such as in [105]. These strategies can also be easily extended to distributed settings.

In addition to the above, though this thesis discusses the accelerated quasi-Newton methods in the context of neural network training, it is not limited to just neural networks but can also be explored in various other fields and applications where optimization algorithms can be applied.

A

Appendix

A.1 Two-loop recursion

Algorithm A.1 Direction Update - Two-loop Recursion

Require: current gradient $\nabla E(\theta_k)$, memory size m, curvature pair $(\sigma_{k-i}, \gamma_{k-i})$ $\forall i = 1, 2, ..., min(k-1, m)$ where σ_k is the difference of current and previous weight vector and γ_k is the difference of current and previous gradient vector

$\eta_k = -\nabla E(\theta_k)$
for $i := 1, 2, ..., \min(m, k-1)$ **do**
    $\beta_i = (\sigma_{k-i}^{\mathrm{T}} \eta_k)/(\sigma_{k-i}^{\mathrm{T}} \gamma_{k-i})$
    $\eta_k = \eta_k - \beta_i \gamma_{k-i}$
end for
if $k > 1$ **then**
    $\eta_k = \eta_k(\sigma_k^{\mathrm{T}} \gamma_k / \gamma_k^{\mathrm{T}} \gamma_k)$
end if
for $i : k - \min(m, (k-1)), \ldots, k-1, k$ **do**
    $\tau = (\gamma_i^{\mathrm{T}} \eta_k)/(\gamma_i^{\mathrm{T}} \sigma_i)$
    $\eta_k = \eta_k - (\beta_i - \tau)\sigma_i$
end for
return η_k

A.2 Sherman Morrison Woodbury Formula

In linear algebra, the Sherman-Morrison formula [110] is used to compute the inverse of the sum of an invertible matrix $\mathbf{A}$ and outer product of the vectors $\mathbf{u}$ and $\mathbf{v}^{\mathrm{T}}$. The formula states that suppose matrix $\mathbf{A} \in \mathbf{R}^{n \times n}$ is an invertible matrix and vectors $\mathbf{u}$ and $\mathbf{v} \in \mathbf{R}^n$, then $\mathbf{A} + \mathbf{u}\mathbf{v}^{\mathrm{T}}$ is invertible iif $1 + \mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{u} \neq 0$. In other words, it gives the inverse of the rank-1 modification of the matrix $\mathbf{A}$ as shown below:

$$\left(\mathbf{A} + \mathbf{u}\mathbf{v}^{\mathrm{T}}\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}}{1 + \mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{u}} \tag{A.1}$$

The Sherman-Morrison-Woodbury formula [111] is the generalization to a rank-k modification of the matrix $\mathbf{A}$ as shown below:

$$\left(\mathbf{A} + \mathbf{U}\mathbf{V}^{\mathrm{T}}\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I} + \mathbf{V}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^{\mathrm{T}}\mathbf{A}^{-1} \tag{A.2}$$

B

List of Publications

A. Academic article related to the qualification

1. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "Accelerating Symmetric Rank-1 Quasi-Newton Method with Nesterov's Gradient for Training Neural Networks", *Algorithms* 2022, 15(1), 6; [Link]

2. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "A Nesterov's Accelerated quasi-Newton method for Global Routing using Deep Reinforcement Learning", *NOLTA Journal*, Vol. 12(3), pp. 323-335, IEICE, Jul 2021 (*Invited Paper*) [Link]

3. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, H. Asai, "aSNAQ: An Adaptive Stochastic Nesterov Accelerated Quasi Newton Method for Training RNNs", *NOLTA Journal,* Vol.E11-N, No.4, pp. 409-421, IEICE Oct. 2020 (*Invited Paper*) [Link]

B. Academic article related to dissertation (including unpublished paper or proceeding) other than listed above

1. S. Indrapriyadarsini, H. Ninomiya, M. Nishimura, "On the Convergence of Stochastic Accelerated quasi-Newton Methods for Neural Networks", 2022 (*under review*).

2. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "A Stochastic Momentum Accelerated Quasi-Newton Method for Neural Networks (Student Abstract)", *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, Feb 2022. [Link]

3. S. Indrapriyadarsini, H. Ninomiya, T. Kamio, H. Asai, "On the Practical Robustness of the Nesterov's Accelerated Quasi-Newton Method", *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, Feb 2022 [Link]

4. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "A modified limited memory Nesterov's accelerated quasi-Newton", NOLTAソサイエティ大会, IEICE, Jun 2021. [Link]

5. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "A Nesterov's Accelerated quasi-Newton method for Global Routing using Deep Reinforcement Learning," *International Symposium on Nonlinear Theory and its Applications*, IEICE, pp. 251-254, Nov 2020 (*Student Paper Award*) [Link]

6. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "A Neural Network Approach to Analog Circuit Design Optimization Using Nesterov's Accelerated Quasi-Newton Method", *Proc. International Symposium on Circuits and Systems (ISCAS)*, IEEE 2020 [Link]

7. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "Neural Networkを用いたAnalog回路設計", 総合大会, IEICE, March 2020

8. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, H. Asai, "An Adaptive Stochastic Nesterov Accelerated Quasi Newton Method for Training RNNs," *International Symposium on Nonlinear Theory and its Applications*, IEICE, pp. 208-211, Dec 2019 (*Best Student Paper Award*) [Link]

9. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, H. Asai, "A Stochastic Quasi-Newton Method with Nesterov's Accelerated Gradient", *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, ECML-PKDD, LNCS vol.11906, pp. 743-760, Springer, Cham, Sept 2019 [Link]

10. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, H. Asai, "Implementation of a modified Nesterov's Accelerated quasi-Newton method on Tensorflow" *Proc. 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018. [Link]

C. Other articles

1. S. Yasuda, S. Indrapriyadarsini, H. Ninomiya, T. Kamio, H. Asai, "addHessian: Combining quasi-Newton method with first-order method for neural network training", *NOLTA Journal*, Vol 13(2), pp. 361-366, IEICE, April 2022 [Link]

2. S. Mahboubi, R. Yamatomi, S. Indrapriyadarsini, H. Ninomiya, H. Asai, "On the study of Memory-Less quasi-Newton Method with Momentum Term for Neural Network Training", *NOLTA Journal*, Vol 13(2), pp. 271-276, IEICE, April 2022 [Link]

3. S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, H. Asai, "Momentum Acceleration of quasi-Newton based Optimization Technique for Neural Network Training", *NOLTA Journal*, Vol. 12(3), pp. 554-574, IEICE, Jul2021 [Link]

4. S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, H. Asai, "A Robust quasi-Newton Training with Adaptive Momentum for Microwave Circuit Models in Neural Networks", *Journal of Signal Processing* 24.1 (2020): 11-17. [Link]

5. S. Yasuda, S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, H. Asai, "A Stochastic Variance Reduced Nesterov's Accelerated Quasi-Newton Method" *Proc. 18th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2019 [Link]

6. S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, H. Asai, "Momentum acceleration of quasi-Newton Training for Neural Networks", *16th Pacific Rim International Conference on Artificial Intelligence*, PRICAI 2019, (pp. 268-281). Springer, Cham. [Link]

D. Presentation in academic conferences

1. S. Indrapriyadarsini, H. Ninomiya, M. Nishimura, "Stochastic Accelerated Quasi-Newton for Training Neural Networks," Eastern European Machine Learning Summer School, EEML Jul 2022 (*Poster*) [Link]

2. S. Indrapriyadarsini, H. Ninomiya, M. Nishimura, "Accelerated Stochastic Quasi-Newton Methods for Training Neural Networks," 8th International Symposium towards Future Advanced Research (ISFAR), Shizuoka University, Mar 2022 (*Best Presentation Award*)

3. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "Accelerating Symmetric Rank 1 Quasi-Newton Method with Nesterov's Gradient", Affinity Workshop WiML NeurIPS, Dec 2021 (*Poster*) [Link]

4. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "A modified limited memory Nesterov's accelerated quasi-Newton", WiML Workshop, ICML Jul 2021(*Poster*) [Link]

5. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "VLSI Physical Design Automation using Deep Reinforcement Learning", WiML Workshop @ NeurIPS, Dec 2020. (*Poster*) [Link]

6. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, H. Asai, "Applications of Deep Learning in Electronic Design Automation", Eastern European Machine Learning Summer School, EEML Jul 2020. (*Video*) [Link]

7. S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, H. Asai, "A Modified Nesterov's Accelerated quasi-Newton Method on Tensorflow," 5th International Symposium towards Future Advanced Research (ISFAR), Shizuoka University, Mar 2019 (*Poster*)

Bibliography

[1] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer Series in Operations Research. Springer, second edition, 2006.

[2] C. C. Aggarwal, Aggarwal, and Lagerstrom-Fife, *Linear algebra and optimization for machine learning*. Springer, 2020.

[3] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *international conference on machine learning*, pp. 2847–2854, PMLR, 2017.

[4] S. Arora, N. Cohen, and E. Hazan, "On the optimization of deep networks: Implicit acceleration by overparameterization," in *International Conference on Machine Learning*, pp. 244–253, PMLR, 2018.

[5] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in neural information processing systems*, pp. 315–323, 2013.

[6] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. Smola, "Stochastic variance reduction for nonconvex optimization," in *International conference on machine learning*, pp. 314–323, PMLR, 2016.

[7] A. Lucchi, B. McWilliams, and T. Hofmann, "A variance reduced stochastic newton method," *arXiv preprint arXiv:1503.08316*, 2015.

[8] H. Jia, X. Zhang, J. Xu, W. Zeng, H. Jiang, X. Yan, and J.-R. Wen, "Variance reduction for deep q-learning using stochastic recursive gradient," *arXiv preprint arXiv:2007.12817*, 2020.

[9] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[10] L. Bottou and Y. L. Cun, "Large scale online learning," in *Advances in neural information processing systems*, pp. 217–224, 2004.

[11] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[12] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[13] J. Konečnỳ and P. Richtárik, "Semi-stochastic gradient descent methods," *Frontiers in Applied Mathematics and Statistics*, vol. 3, p. 9, 2017.

[14] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," *Advances in neural information processing systems*, vol. 27, 2014.

[15] L. M. Nguyen, J. Liu, K. Scheinberg, and M. Takáč, "Sarah: A novel method for machine learning problems using stochastic recursive gradient," in *International Conference on Machine Learning*, pp. 2613–2621, PMLR, 2017.

[16] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, no. 1, pp. 83–112, 2017.

[17] R. Bollapragada, R. Byrd, and J. Nocedal, "Adaptive sampling strategies for stochastic optimization," *SIAM Journal on Optimization*, vol. 28, no. 4, pp. 3312–3343, 2018.

[18] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.

[19] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning.," *ICML (3)*, vol. 28, no. 1139-1147, p. 5, 2013.

[20] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate o(1/kˆ2)," in *Dokl. akad. nauk Sssr*, vol. 269, pp. 543–547, 1983.

[21] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop, coursera: Neural networks for machine learning," *University of Toronto, Technical Report*, 2012.

[22] D. P. Kingma and J. Ba, "Adam : A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," *arXiv preprint arXiv:1904.09237*, 2019.

[24] T. Dozat, "Incorporating nesterov momentum into adam," *Workshop track-ICLR*, 2016.

[25] C. Gulcehre, M. Moczulski, and Y. Bengio, "Adasecant: robust adaptive secant method for stochastic gradient," *arXiv preprint arXiv:1412.7419*, 2014.

[26] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," *Advances in neural information processing systems*, vol. 30, 2017.

[27] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, and H. Asai, "Implementation of a modified nesterov's accelerated quasi-newton method on tensorflow," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1147–1154, IEEE, 2018.

[28] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, and H. Asai, "A modified limited memory nesterov's accelerated quasi-newton," *arXiv preprint arXiv:2112.01327*, 2021.

[29] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, and H. Asai, "A neural network approach to analog circuit design optimization using nesterov's accelerated quasi-newton method," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–1, IEEE, 2020.

[30] Y. Nesterov *et al.*, *Lectures on convex optimization*, vol. 137. Springer, 2018.

[31] H. Ninomiya, "A novel quasi-newton-based optimization for neural network training incorporating nesterov's accelerated gradient," *Nonlinear Theory and Its Applications, IEICE*, vol. 8, no. 4, pp. 289–301, 2017.

[32] S. Mahboubi and H. Ninomiya, "A novel training algorithm based on limited-memory quasi-newton method with nesterov's accelerated gradient in neural networks and its application to highly-nonlinear modeling of microwave circuit," *IARIA International Journal on Advances in Software*, vol. 11, no. 3-4, pp. 323–334, 2018.

[33] S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, and H. Asai, "Momentum acceleration of quasi-newton training for neural networks," in *Pacific Rim International Conference on Artificial Intelligence*, pp. 268–281, Springer, 2019.

[34] S. Full-wave, "3d planar electromagnetic field solver software for high frequency em simulation, sonnet software."

[35] A. S. Sedra, K. C. Smith, T. C. Carusone, and V. Gaudet, *Microelectronic circuits*, vol. 4. Oxford university press New York, 2004.

[36] Z. Wang, X. Luo, and Z. Gong, "Application of deep learning in analog circuit sizing," in *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, pp. 571–575, 2018.

[37] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, and H. Asai, "A stochastic quasi-newton method with nesterov's accelerated gradient," in *ECML-PKDD*, Springer, 2019.

[38] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, and H. Asai, "A stochastic momentum accelerated quasi-newton method for neural networks (student abstract)," in *Proceedings of the 36th AAAI Conference on Artificial Intelligence*, 2022.

[39] X. Peng, L. Li, and F.-Y. Wang, "Accelerating minibatch stochastic gradient descent using typicality sampling," *arXiv preprint arXiv:1903.04192*, 2019.

[40] J. Martens, "Deep learning via hessian-free optimization.," in *ICML*, vol. 27, pp. 735–742, 2010.

[41] F. Roosta-Khorasani and M. W. Mahoney, "Sub-sampled newton methods i: globally convergent algorithms," *arXiv preprint arXiv:1601.04737*, 2016.

[42] J. E. Dennis, Jr and J. J. Moré, "Quasi-newton methods, motivation and theory," *SIAM review*, vol. 19, no. 1, pp. 46–89, 1977.

[43] N. Agarwal, B. Bullins, and E. Hazan, "Second-order stochastic optimization in linear time," *stat*, vol. 1050, p. 15, 2016.

[44] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-newton method for online convex optimization," in *Artificial Intelligence and Statistics*, pp. 436–443, 2007.

[45] A. Mokhtari and A. Ribeiro, "Res: Regularized stochastic bfgs algorithm," *IEEE Transactions on Signal Processing*, vol. 62, no. 23, pp. 6089–6104, 2014.

[46] A. Mokhtari and A. Ribeiro, "Global convergence of online limited memory bfgs," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 3151–3181, 2015.

[47] R. H. Byrd, S. L. Hansen, J. Nocedal, and Y. Singer, "A stochastic quasi-newton method for large-scale optimization," *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1008–1031, 2016.

[48] X. Wang, S. Ma, D. Goldfarb, and W. Liu, "Stochastic quasi-newton methods for nonconvex stochastic optimization," *SIAM Journal on Optimization*, vol. 27, no. 2, pp. 927–956, 2017.

[49] Y. Li and H. Liu, "Implementation of stochastic quasi-newton's method in pytorch," *arXiv preprint arXiv:1805.02338*, 2018.

[50] P. Moritz, R. Nishihara, and M. Jordan, "A linearly-convergent stochastic l-bfgs algorithm," in *Artificial Intelligence and Statistics*, pp. 249–258, 2016.

[51] R. Bollapragada, D. Mudigere, J. Nocedal, H.-J. M. Shi, and P. T. P. Tang, "A progressive batching l-bfgs method for machine learning," *arXiv preprint arXiv:1802.05374*, 2018.

[52] R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal, "On the use of stochastic hessian information in optimization methods for machine learning," *SIAM Journal on Optimization*, vol. 21, no. 3, pp. 977–995, 2011.

[53] R. Gower, D. Goldfarb, and P. Richtárik, "Stochastic block bfgs: Squeezing more curvature out of data," in *International Conference on Machine Learning*, pp. 1869–1878, 2016.

[54] L. Zhang, "A globally convergent bfgs method for nonconvex minimization without line searches," *Optimization Methods and Software*, vol. 20, no. 6, pp. 737–747, 2005.

[55] Y.-H. Dai, "Convergence properties of the bfgs algoritm," *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 693–701, 2002.

[56] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 928–936, 2003.

[57] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online] Available: http://yann.lecun.com/exdb/mnist*, 2010.

[58] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547–553, 2009.

[59] E. Alpaydin and C. Kaynak, "Optical recognition of handwritten digits data set," *UCI Machine Learning Repository*, 1998.

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

[61] M. J. Powell, "Some global convergence properties of a variable metric algorithm for minimization without exact line searches," in *Nonlinear programming, SIAM-AMS proceedings*, vol. 9, 1976.

[62] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *ICML*, pp. 1310–1318, 2013.

[63] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, and H. Asai, "An adaptive stochastic nesterov's accelerated quasi-newton method for training rnns," in *International Symposium on Nonlinear Theory and Its Applications, NOLTA'19*, The Institute of Electronics, Information and Communication Engineers, IEICE, 2019.

[64] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, and H. Asai, "asnaq: An adaptive stochastic nesterov's accelerated quasi-newton method for training rnns," *Nonlinear Theory and Its Applications, IEICE*, vol. 11, no. 4, pp. 409–421, 2020.

[65] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[66] I. Sutskever, *Training recurrent neural networks*. University of Toronto, Ontario, Canada, 2013.

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

[68] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th ICML*, pp. 1033–1040, 2011.

[69] C.-C. Peng and G. D. Magoulas, "Nonmonotone bfgs-trained recurrent neural networks for temporal sequence processing," *Applied mathematics and computation*, vol. 217, no. 12, pp. 5421–5441, 2011.

[70] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[71] N. S. Keskar and A. S. Berahas, "adaqn: An adaptive quasi-newton algorithm for training rnns," in *Joint ECML-KDD*, pp. 1–16, Springer, 2016.

[72] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[73] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[74] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, and H. Asai, "A nesterov's accelerated quasi-newton method for global routing using deep reinforcement learning," in *International Symposium on Nonlinear Theory and Its Applications, NOLTA'20*, pp. 251–254, November 2020.

[75] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, K. Takeshi, and H. Asai, "A nesterov's accelerated quasi-newton method for global routing using deep reinforcement learning," *Nonlinear Theory and Its Applications, IEICE*, vol. 12, no. 3, pp. 323–335, 2021.

[76] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1st ed., 1998.

[77] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, May 1992.

[78] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, February 2015.

[79] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in 30^{th} *AAAI Conf. on Artificial Intelligence*, March 2016.

[80] S. Ghiassian, B. Rafiee, Y. L. Lo, and A. White, "Improving performance in reinforcement learning by breaking generalization in neural networks," in *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).*, May 2020.

[81] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. Thirty-Second AAAI Conference on Artificial Intelligence*, April 2018.

[82] A. Jacobsen, M. Schlegel, C. Linke, T. Degris, A. White, and M. White, "Meta-descent for online, continual prediction," in *Proc. AAAI Conference on Artificial Intelligence*, 2019.

[83] W.-Y. Zhao, X.-Y. Guan, Y. Liu, X. Zhao, and J. Peng, "Stochastic variance reduction for deep q-learning," *arXiv preprint arXiv:1905.08152*, 2019.

[84] G. W. Clow, "A global routing algorithm for general cells," in *21st Design Automation Conference Proceedings*, pp. 45–51, IEEE, June 1984.

[85] W. T. J. Chan, P. H. Ho, A. B. Kahng, and P. Saxena, "Routability optimization for industrial designs at sub-14nm process nodes using machine learning," in *Proc. of the 2017 ACM on International Symposium on Physical Design*, pp. 15–21, March 2017.

[86] B. Li and P. D. Franzon., "Machine learning in physical design," in *Proc. IEEE 25th Conference on Electrical Performance Of Electronic Packaging And Systems*, pp. 147–150, IEEE, October 2016.

[87] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint*, December 2013.

[88] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint*, September 2015.

[89] H. Liao, W. Zhang, X. Dong, B. Poczos, K. Shimada, and L. Burak Kara, "A deep reinforcement learning approach for global routing," *Journal of Mechanical Design*, vol. 142, no. 6, 2020.

[90] H.-Y. Chen and Y.-W. Chang, "Global and detailed routing," in *Electronic Design Automation*, pp. 687–749, Elsevier, 2009.

[91] P. G. Mehta and S. P. Meyn, "Convex q-learning, part 1: Deterministic optimal control," *arXiv preprint arXiv:2008.03559*, 2020.

[92] S. Indrapriyadarsini, S. Mahboubi, H. Ninomiya, T. Kamio, and H. Asai, "Accelerating symmetric rank-1 quasi-newton method with nesterov's gradient for training neural networks," *Algorithms*, vol. 15, no. 1, p. 6, 2021.

[93] R. H. Byrd, H. F. Khalfan, and R. B. Schnabel, "Analysis of a symmetric rank-one trust region method," *SIAM Journal on Optimization*, vol. 6, no. 4, pp. 1025–1039, 1996.

[94] J. Brust, J. B. Erway, and R. F. Marcia, "On solving l-sr1 trust-region subproblems," *Computational Optimization and Applications*, vol. 66, no. 2, pp. 245–266, 2017.

[95] P. Spellucci, "A modified rank one update which converges q-superlinearly," *Computational Optimization and Applications*, vol. 19, no. 3, pp. 273–296, 2001.

[96] F. Modarres, M. A. Hassan, and W. J. Leong, "A symmetric rank-one method based on extra updating techniques for unconstrained optimization," *Computers & Mathematics with Applications*, vol. 62, no. 1, pp. 392–400, 2011.

[97] H. F. Khalfan, R. H. Byrd, and R. B. Schnabel, "A theoretical and experimental study of the symmetric rank-one update," *SIAM Journal on Optimization*, vol. 3, no. 1, pp. 1–24, 1993.

[98] M. Jahani, M. Nazari, S. Rusakov, A. S. Berahas, and M. Takáč, "Scaling up quasi-newton algorithms: Communication efficient distributed sr1," in *International Conference on Machine Learning, Optimization, and Data Science*, pp. 41–54, Springer, 2020.

[99] A. Berahas, M. Jahani, P. Richtarik, and M. Takáč, "Quasi-newton methods for machine learning: forget the past, just sample," *Optimization Methods and Software*, pp. 1–37, 2021.

[100] B. O'donoghue and E. Candes, "Adaptive restart for accelerated gradient schemes," *Foundations of computational mathematics*, vol. 15, no. 3, pp. 715–732, 2015.

[101] S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya, H. Asai, *et al.*, "Momentum acceleration of quasi-newton based optimization technique for neural network training," *Nonlinear Theory and Its Applications, IEICE*, vol. 12, no. 3, pp. 554–574, 2021.

[102] R. H. Byrd, J. Nocedal, and R. B. Schnabel, "Representations of quasi-newton matrices and their use in limited memory methods," *Mathematical Programming*, vol. 63, no. 1, pp. 129–156, 1994.

[103] X. Lu and R. H. Byrd, *A Study of the Limited Memory Sr1 Method in Practice*. PhD thesis, University of Colorado at Boulder, USA, 1996.

[104] G. A. Shultz, R. B. Schnabel, and R. H. Byrd, "A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties," *SIAM Journal on Numerical analysis*, vol. 22, no. 1, pp. 47–67, 1985.

[105] K. Crammer, A. Kulesza, and M. Dredze, "Adaptive regularization of weight vectors," *Advances in neural information processing systems*, vol. 22, 2009.

[106] J. Ba, R. Grosse, and J. Martens, "Distributed second-order optimization using kronecker-factored approximations," 2016.

[107] R. Anil, V. Gupta, T. Koren, K. Regan, and Y. Singer, "Scalable second order optimization for deep learning," *arXiv preprint arXiv:2002.09018*, 2020.

[108] W. Chen, Z. Wang, and J. Zhou, "Large-scale l-bfgs using mapreduce," *Advances in neural information processing systems*, vol. 27, 2014.

[109] Y. Fei, G. Rong, B. Wang, and W. Wang, "Parallel l-bfgs-b algorithm on gpu," *Computers & graphics*, vol. 40, pp. 1–9, 2014.

[110] J. Sherman and W. J. Morrison, "Adjustment of an inverse matrix corresponding to a change in one element of a given matrix," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[111] M. A. Woodbury, *Inverting modified matrices*. Statistical Research Group, 1950.