Journal of Theoretical and Applied Information Technology
30th November 2021. Vol.99. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DEEP ARTIFICIAL NEURAL NETWORKS OPTIMIZATIONS USING SECOND-ORDER NEWTON'S METHOD

[1]RAWAN GHNEMAT, [2]AHMAD GHANNAM, [3]ABRAR M. AL-SOWI

[1]Professor, Princess Sumaya University for Technology, Computer Science Department, Jordan
[2]Princess Sumaya University for Technology, Computer Science Department, Jordan
[3]The University of Jordan, Computer Information Systems Department, Jordan

E-mail: [1]r.ghnemat@psut.edu.jo, [2]ahm20178072@std.psut.edu.jo, [3]abrar.sowi@gmail.com

ABSTRACT

Optimization is the process of finding parameters that return the maximum or minimum value of a function, where the function symbolizes the effort needed or the desired benefit. First-order stochastic gradient (SG) methods are often used to solve deep learning models that involve a hard non-convex optimization problem. Although second-order methods can ensure faster convergence, they have been less explored because processing time and costs are high. Optimizing deep learning models is a challenging problem; many deep learning companies spend a lot of their resources on training deep models. This paper proposes an implementation and evaluation of Newton's second-order optimization method, Hessian Free Optimization (HFO), on fully connected feed-forward networks, and enhances the method by the integration with some acceleration techniques such as Momentum and Root Mean Square Propagation (RMSProp). The paper also proposed a hybrid algorithm capable of combining two-degree orders, first-order, and second-order optimization methods. The hybrid algorithm can achieve better convergence (5% better in testing loss) compared to first-order methods with approximately the same time consumption.

Keywords: *Machine Learning, Optimization Method, Deep Neural Network, Newton's Method, Deep Learning Models, Hessian Free Optimization (HFO).*

1. INTRODUCTION

Recently, machine learning and deep learning have grown in quick paces. This improves the development of different theoretical and applied fields. Solving machine learning and deep learning models, the artificial neural network models and their different forms, such as feed-forward neural networks, recurrent neural networks, and conventional neural networks, usually require a high amount of numerical computation for achieving their minimum value when computing the loss function and updating the models' parameters in every step, and by going back to apply the same computation several times to reach the desired output. This iterative process is called the optimization process.

Optimization process solves mathematical problems in iterative steps to update estimates of the solution, instead of finding analytically a short form formula that provides a symbolic expression of the solution. The common operations include optimization which is finding the value/conditions of an argument that minimizes or maximizes a function and solving systems of linear equations [1].

The optimization process in the neural networks is vital because it is highly correlated with the model's performance, where the optimization leads to models that perform better. Overall, the difference between a neural network model that performs well and another that performs poorly is due to the quality of the optimization.

Optimization of artificial neural networks is considered the most critical step in the developing phase, in which the training process goes through two significant steps, first the feed-forward propagation wherein this step, the network is fed the input features from the training samples and passes

the computation results to the output layer through some hidden-layers, the computation and activation functions, and then comparing the final output with the actual one to compute the error for the network, second the back-propagation wherein this step the network goes back to every layer to adjust the weights based on the error, trying to minimize it to make the network's prediction better in every iteration.

Optimizing deep learning models suffers from some challenges, such as vanishing gradients. In previous publications, Optical Back Propagation (OBP) algorithm [2] was introduced to overcome the problem of vanishing gradients of non-linear activation functions such as Sigmoid and Tanh. In numerical analysis, Newton's method [3], or Newton–Raphson method, is a root-finding algorithm that computes consecutively better approximations to the roots (or zeros) of a real-valued function.

Optimization algorithms can be categorized in terms of their use of momentum and/or root mean squared, which are called acceleration techniques and can be integrated with the optimizers to enhance the convergence at a faster speed to the goal.

Gradient-Based optimization is one of the most dominant techniques for optimization; it is used almost in optimizing most of the deep learning models. It can minimize the error very fast due to the low cost of computation of the first derivative. However, it is not always accurate when compared to higher-order derivatives. Gradient-based optimization is found very efficient in training neural network models, which resulted in breakthroughs in applications such as face recognition [4], speech recognition [5], and natural language processing [6]. It is derived from Newton's method, and it adds a constant value to the optimization function, which is called the learning rate, and it does not rely on the second derivative of the targeted function.

The usage of machine learning and deep learning is increasing in this revolutionary time in all fields. Consequently, the importance of having accurate models requires powerful optimization methods, training data, and of course, having a meaningful objective function as well. In machine learning and deep learning, exceedingly hard non-convex optimization problems have appeared, which are often solved by first-order stochastic methods (SG).

This paper focuses on the optimization function, a second-order Newton's method that will provide faster and accurate convergence because of the high and more critical information that can be obtained from computing the second-order derivative.

Developing deep learning models is the process of finding the parameters that yield the best performance. Finding the parameters is known as training, and those parameters are found using an optimization process. It is well known that the performance of deep learning depends on the size of the training data; the more extensive set of data is fed, the better the model is given. Machine learning practitioners are used to say that "a dumb model with more data is better than a smart model with less amount of it".

It is a time-consuming process to train deep learning models or networks over several iterations, where for every iteration, it may take several hours or days. Hence enhancing the efficiency of the training is very important for deep learning.

With the specific requirements and nature of deep learning models' applications, the need to develop accurate and efficient deep learning models arises. As a representative of first-order optimization methods, the gradient descent method has been used widely in recent years and is developing at an exceedingly high pace. However, the characteristics or application scope of these methods have not been paid attention to by users. Second-order optimization methods, compared with first-order optimization methods, converge rapidly in which the search direction is made more efficient by the curvature information. These methods attract far-flung attention, but they face some challenges. The obstacles in second-order methods exist in the computation time and storing cost to handle the Hessian matrix's inverse matrix. Approximations of Newton's method have been developed to solve this

problem. The approximation of the Hessian matrix via some techniques [7] [8] is considered a solution for most of them.

This paper aims to improve the efficiency of training deep learning models by decreasing the number of required iterations to achieve the desired output of the models.

The proposed method aims to reduce the number of iterations by implementing a second-order Newton's method to optimize the neural network. This work also aims at investigating the integration between the acceleration techniques with second-order Newton's method.

The main contribution to this thesis is the proposal of several acceleration techniques for deep learning optimization algorithms, which surpass traditional optimization techniques and result in faster convergence time and more intelligent agents. The main contributions in this research are given below:

- Integrating Hessian Free Optimization with momentum to accelerate the optimization of a deep learning algorithm.
- Integrating Hessian Free Optimization with Root Mean Square Propagation (RMSProp) to accelerate the optimization of a deep learning algorithm.
- Proposing a novel technique combining first-order and second-order methods to be named Hybrid Hessian Free Optimization (HHFO).
- Evaluation of the proposed optimization techniques and demonstrating the superiority of the combination of two order optimization methods.

The rest of the paper is organized as follows: Section 2 provides a brief background and related works. In Section 3, we present the research methodology in detail. Section 4 shows the experimental results with the evaluation and discussion. Finally, in section 5, we conclude our work and identify future work avenues

2. BACKGROUND AND RELATED WORKS

The development cycle of machine learning is outlined in three critical phases. These phases are designed to define the model architecture, identify the loss function, and overcome the minimum loss function to determine the model's parameters. Machine learning modelling architecture is the first two phases, while the last phase is to find an answer to the coveted model through some optimization techniques [9]. In this section a comprehensive related work is presented, optimization methods are introduced: First-order methods, second order methods and Hessian free method.

2.1. Optimization Methods

From the gradient-based information point of view, optimization methods are classified into first-order optimization methods, high-order optimization methods, which includes second-order methods. Such techniques have a long journey and are continuously developing, and in various applications, they are improving, and they perform well. For more details on the equations in this section, the reader is referred to [3], [9].

2.1.1. First-order methods

In the machine learning field, gradient descent-based approaches are the most widely used first-order optimizations. In this section, some gradient descent methods, along with the algorithms' development, are introduced.

The Gradient Descent (GD) approach is one of the first and most popular methods for optimization. The principle of this approach is to change variables in the opposite direction of the objective function's gradients iteratively. The updated variables are used to converge gradually to the optimal value of the objective function. For every iteration, the training speed defines the step size and consequently affects the number of steps to achieve the optimum value. A well-known algorithm is the steepest descent algorithm. The steepest descent algorithm aims to choose an appropriate search path in every iteration so that the objective function's value minimizes

extremely fast. Steepest descent and gradient descent and are different due to the negative gradient direction does not always descend as fast as possible. An example of the steepest descent is gradient descent using the Euclidean norm. First, we give the gradient descent method's formal expression.

For the simplest machine learning model, the linear regression function, let $f_\theta(x)$ be the desired model to be trained, $L(\theta)$ is the loss function for the model, and θ is the function's parameter to be optimized for minimizing the loss function.

$$L(\theta) = \frac{1}{2N}\sum_{i=1}^{N}(y^i - f_\theta(x^i))^2 \quad (1)$$

$$f_\theta(x) = \sum_{j=1}^{D}\theta_j x_j \quad (2)$$

where N is the amount of training samples, D is the input dimension size, x^i is an input variable, and y^i is the output target. The gradient descent performs iteratively the two steps below until a satisfied convergence occurs.

First, as shown in equation $\frac{\partial L(\theta)}{\partial \theta_j} = -\frac{1}{N}\sum_{i=1}^{N}(y^i - f_\theta(x^i))x_j^i$ (3), deriving $L(\theta)$ for θ_j to compute the gradient for every θ_j. Then, as in equation $\acute{\theta}_j = \theta_j + \eta \cdot \frac{1}{N}\sum_{i=1}^{N}(y^i - f_\theta(x^i))x_j^i$ (4), updating every θ_j in the opposite gradient direction for minimizing the error function:

$$\frac{\partial L(\theta)}{\partial \theta_j} = -\frac{1}{N}\sum_{i=1}^{N}(y^i - f_\theta(x^i))x_j^i \quad (3)$$

$$\acute{\theta}_j = \theta_j + \eta \cdot \frac{1}{N}\sum_{i=1}^{N}(y^i - f_\theta(x^i))x_j^i \quad (4)$$

The gradient descent method is simply to be implemented and used since it is derived from Newton's Raphson Method. Also, it immediately manipulates the first-order derivative of the loss function, which is easy to compute in almost all objective functions. If the objective function is convex, the solution is considered as global optimal. Although it is easy to implement when dealing with large-scale data, the cost is high as all training samples are used on every iteration step.

Batch gradient descent is another name of the gradient descent. Consequently, the mini-batch gradient descent and stochastic gradient descent methods arise. In the batch gradient descent, $O(ND)$ is the computational complexity for every iteration. This is expensive for every iteration and this method does not allow online updates. Consequently, several methods of parallelization have been suggested to minimize the expense of calculation and stochastic gradient descent (SGD) was introduced [10] [3].

Stochastic gradient descent (SGD) uses one random sample for updating the gradient iteratively instead of calculating the gradient value for all training samples at once. Consequently, the cost of the SGD algorithm is independent of the number of samples and can reach a sublinear convergence computing time [11]. SGD shortens the time required for updating the gradient when dealing with large-scale data and eliminates a particular amount of redundant computations, which makes the process faster significantly. The method can reach an optimum convergence speed in the sense of a strong convex problem [12], [13], [14], [15].

SGD overcomes a batch gradient descent since the latter cannot be utilized for online learning. SGD loss function is expressed as the following:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y^i - f_\theta(x^i))^2 = \frac{1}{N}\sum_{i=1}^{N} cost(\theta, (x^i, y^i)) \quad (5)$$

When SGD selects a random sample i the loss function will be $L^*(\theta)$:

$$L^*(\theta) = cost\left(\theta, (x^i, y^i)\right) = \frac{1}{2}(y^i - f_\theta(x^i))^2 \quad (6)$$

The SGD updates the gradient by using a random training sample i in every iteration, rather than all training samples.

$$\acute{\theta} = \theta + \eta(y^i - f_\theta(x^i))x^i \quad (7)$$

Because SGD requires just one sample for every iteration, the computational complexity is $O(D)$ per iteration, where D is the number of features. When the number of samples N is high, the update frequency for every iteration of SGD is much faster than that of the batch gradient descent. SGD

increases overall optimization performance for every iteration, but this is negligible compared to the high number of iterations caused by many samples.

In general, only a small number of samples can be used to get an optimal solution even if the sample size is considerable. As a result, SGD can reduce the computational complexity and accelerate the speed of convergence compared to batch methods. However, one difficulty in SGD is that due to the additional noise caused by random selection, the gradient path oscillates, and the search method becomes blind in the space of the answer. Unlike the batch gradient descent that always moves over the opposite direction of the gradient towards the optimal value, the gradients' variance that is created in SGD is large, and the movement's direction in SGD is biased. The mini-batch gradient descent method (MSGD) was introduced as a compromise between the two methods (GD and SGD).

As the test involves parameter changes in every iteration, the Mini-Batch Gradient Descent (MSGD) uses c independent, equally distributed samples (c is usually between 64 and 256 [16]). It reduces gradient variance and stabilizes convergence, which increases optimization rate. Mini-Batch Gradient Descent has a better opportunity of exploring an optimal global solution in case of complex problems as it allows the target function to reach a new minimum. However, MSGD is always fluctuating, which can slow down the convergence process.

Other information concerning the use of MSGD in the concrete optimization process is yet to be noted [16], such as the selection of an appropriate learning rate. A too low rate of learning will lead to a slower rate of convergence, while an overly high rate of learning will impede convergence and reduce loss function. To resolve the problem, a predefined list of study levels or a certain threshold can be identified and the learning rate changed during the test process [17], [18]. Such lists or thresholds must however be specified in advance based on the data set features. It is also unacceptable for all variables to use the same learning rate. When there are fragmented data and features at different frequencies, the resulting parameters are not supposed to be modified with the same learning rate. With features that are less common, a higher learning rate is often expected [19], [20]. In addition to the learning rate, a common challenge is how to avoid the target function getting stuck in constant local minimum numbers. Some experiments have found that this problem originates from the "saddle point" [21] rather than the local minimum values. The slopes around the saddle points are in opposite directions, consequently, these points increase the opportunity for MSGB of being trapped. Some researchers have been conducted to avoid trapping at saddle points [22], [23].

SGD and MSGB are popular and widely used, but they sometimes extend the learning process. It is worth investigating how to improve the optimizers' parameters, such as the learning rate that affects the convergence speed, avoiding being trapped at saddle points during the search. A lot of work is being done to boost SGD and MSGD. For example, it was suggested that the momentum concept could be implemented in both methods [24]. The definition of momentum is inferred from physics mechanics, which describes the objects' inertia. The idea of using momentum in SGD is to retain, to some degree, the effect of the previous update direction on the next iteration. The momentum method will speed convergence in the case of high curvatures, low but stable gradients, or noisy gradients [1].

The variable v is provided by the momentum algorithm as the speed, describing the direction and rate of motion of the parameter in space. The speed is determined as the exponential moving average (EMA), which is a method that applies weighting factors that reduce the average's dependence on the previous points exponentially [25], of the negative gradient. The speed update in gradient descent method is defined as, $v = \eta \cdot \left(-\frac{\partial L(\theta)}{\partial \theta}\right)$ every iteration. The usage of the momentum algorithm takes into consideration the friction factor as it takes the previous update v^{old} multiplied mtm factor, its range between [0, 1] in addition to the speed v which is calculated by gradient descent. The new formulation for the speed is expressed as:

$$v = \eta \cdot \left(-\frac{\partial L(\theta)}{\partial \theta}\right) + v^{old} \cdot mtm \quad (8)$$

Where the momentum factor is mtm. If new gradient has the same direction for prior speed v^{old}, the prior speed will accelerate the search mission. The proper momentum is used to speed up the convergence if a small learning rate is used. If new gradient comes close to 0, the method will continue to update v to achieve balance and be faded away by friction. In the training process it is desirable to break from the local minimum so that search processes are converged faster [24], [26]. If the new gradient direction is contrary to the prior v^{old} direction, the v^{old} value will decelerate this search.

Another important issue is how the learning rate can be determined. The oscillation is more probable if the solution is close to the optimal point. Consequently, the learning rate must be changed. Some research has introduced the learning rate decay method which reduces the learning rate over the iteration period [27]. The learning rate decay formula is defined as:

$$\eta_t = \frac{\eta_0}{1+d \cdot t} \quad (9)$$

Where η_t is the tth iteration learning rate, η_0 is the initial learning rate, and d is the decay factor, its range between [0, 1]. As the equation indicates, the larger the d is, the higher the decay will be. if $d = 0$, the learning rate maintains the same value, likewise, if $d = 1$ the learning rate fades away more quickly.

Manually controlling the learning rate has a major impact on the efficiency of the SGD method. Choosing a proper value of the learning rate is a tricky problem [19], [20], [28]. Several adaptive methods have been introduced to automatically tune the learning rate. Such approaches are variable-free modification, easy to converge, and generally do not produce bad results. They are used extensively in deep neural networks to solve the problem optimizations. AdaGrad [19] is the most obvious enhancement for SGD. In some previous iterations, AdaGrad dynamically tunes the learning rate using information from the previous gradients. The equations for the update are the following:

$$\begin{cases} g_t = \frac{\partial L(\theta_t)}{\partial \theta} \\ V_t = \sqrt{\sum_{i=1}^{t}(g_i)^2 + \epsilon} \\ \theta_{t+1} = \eta \frac{g_t}{V_t} \end{cases} \quad (10)$$

Where g_t is the parameter θ gradient in t iteration, V_t is the cumulative previous gradient of the parameter θ in t iteration, and θ_t is the value of parameter θ in t iteration. AdaGrad differs from the SGD in the learning rate that is no longer predetermined in a parameter update process but computed on all the previous gradients accumulated to current iteration. AdaGrad removes the requirement to manually change the learning rate. The default value of most implementations is 0.01 for η in the equation (10).

Although AdaGrad adapts the learning rate, there are still two problems. 1) The global/initial learning rate still has to be set manually for the algorithm. 2) As the time of training increases, the cumulative gradient becomes very large, making the learning rate tends to zero, resulting in an inefficient adjustment of the parameters. AdaGrad has been further enhanced to AdaDelta [29] and RMSProp [30] in order to solve the issue for the learning rate of recaching a value of zero. The aim is not to accumulate all previous gradients, but to focus only on some gradients inside a window over a period of time and to use the EMA to measure the accumulated momentum of the second order:

$$V_t = \sqrt{\beta V_{t-1} + (1-\beta)(g_t)^2} \quad (11)$$

where β is the parameter of exponential decay. AdaDelta and RMSProp were developed at the same time independently to overcome AdaGrad's dramatically decreasing learning rates which tends to reach a value of zero.

Another innovative SGD tool, Adaptive Moment Estimation (Adam) [20], implements adaptive learning rates on every parameter. It merges momentum methods and the adaptive learning rate together. Besides saving an exponentially decaying average of previous square gradients V_t, similar to RMSProp and AdaDelta, Adam maintains an exponentially decaying average of previous

gradients m_t as well, identical to the momentum method:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad (12)$$

$$V_t = \sqrt{\beta_2 V_{t-1} + (1-\beta_2)(g_t)^2} \quad (13)$$

where β_1 and β_2 are the exponential rate of decay. The final parameter θ update formula is:

$$\theta_{t+1} = m_t - \eta \frac{\sqrt{1-\beta_2}}{1-\beta_1} \frac{m_t}{V_t+\epsilon} \quad (14)$$

It is suggested that the default values of β_1, β_2 and ϵ are set to 0.9, 0.999, and 10^{-8}, respectively. In practice, Adam performs better compared to other adaptive learning-rate algorithms.

Table 2 at the end of this section outlines First-Order Optimization Methods, their features, pros, and cons.

2.1.2. Second-order methods

Second-order optimization methods are derived directly from Newton's method, which the latter is considered a principal optimization method in the numerical analysis field [3]. These optimization methods have been less explored due to the high amount of memory needed to store the hessian information and the cost needed for these computations [31]. Various alternative methods of second-order derivatives are consequently proposed over the years to address these challenges. The additional hessian information makes these methods able to provide a better route towards the curvature of the error surface.

Table 2 at the end of this section outlines Second-Order Optimization Methods, their features, pros, and cons.

Hessian matrices are used to ease the hyperparameter's tuning by adjusting the step size according to the different phases of learning. In the section, some second-order optimization methods are introduced.

Newton's method computes the parameters' updates iteratively by integrating inverse Hessian matrices, $\left(\frac{\partial^2 L(\theta)}{\partial \theta^2}\right)^{-1} = H^{-1}$. The new formulation for Newton's method is expressed as:

$$\theta_{t+1} = \theta_t - \eta \left(\frac{\partial^2 L(\theta)}{\partial \theta^2}\right)^{-1} \frac{\partial L(\theta)}{\partial \theta} \quad (15)$$

Newton's method is extremely powerful. One iteration is only required to reach minimum value [32]. However, the computational cost for every iteration is $O(N^3)$ and requires an enormous amount of memory to save NxN matrices. Consequently, this method is unacceptable to be explicitly applied to deep neural network models that have billions of parameters in critical cases. Consequently, some approximate of inverse Hessian matrices have been proposed to overcome this issue and to be used for actual usage.

Another powerful optimization technique is the CG (Conjugate Gradient) approach, which solves a system of large-scale linear equations, and for non-linear optimization problems [3]. The methods of the first order, as we know, converge slowly despite their simplicity. In general, optimization methods of a higher-order, such as second-order, require high and expensive computations. The CG algorithm comes in the middle between the lower order and the higher-order optimization approaches as it balances between the simplicity from one hand and the fast convergence from the other hand. CG approach has been suggested early in the 1960s to solve a linear equation and replace the standard Gaussian elimination technique [33]. Then, the CG technique was developed further in 1946 to include non-linear functions and generalized optimization techniques such as Polak&Ribiere and Fletcher&Reeves [3].

The linear conjugate gradient method is described for solving the following model. Consider a linear system:

$$A\theta = b \quad (16)$$

where A is an $n \times n$ positive, symmetric-definite matrix. The goal is to find a proper solution of the θ value given that the matrix A and vector b are predefined and known in advanced. The model $A\theta=b$ (16) can be declared equivalently as the following minimization problem, both $A\theta=b$

$$\min_{\theta} F(\theta) = \frac{1}{2}\theta^T A\theta - b\theta + c \quad (17)$$

The conjugate gradient method can be viewed as either an algorithm for solving linear systems or as a technique for minimizing convex quadratic functions. The gradient of $F(\theta)$ and can be obtained simply, and it equals the residual of the linear system [3], that is:

$$r(\theta) = \nabla F(\theta) = A\theta - b \quad (18)$$

A set of non-zero vectors $\{d_1, d_2, d_3, \dots, d_n\}$ is said to be conjugate with respect to A if any two unequal vectors, d_i, d_j are conjugate with respect to A [3], that is:

$$d_i^T A d_j = 0 \quad (19)$$

Consequently, the starting point $\theta_0$can be updated by generating the update sequence $\{\theta_1, \theta_2, \theta_3, \dots, \theta_n\}$ given a set of conjugate directions $\{d_1, d_2, d_3, \dots, d_{n-1}\}$

$$\theta_{t+1} = \theta_t + \eta_t d_t \quad (20)$$

where η_t is the unknown step size and it can be found by any search method. One simple search technique is the linear search where η_t is updated

$$\eta_t = \frac{r_t^{\mathrm{T}} r_t}{d_t^{\mathrm{T}} A d_t} \quad (21)$$

The search direction d_t is calculated by linearly combining the previous search direction d_{t-1} with the negative residual, that is:

$$d_t = -r_t + \beta_t d_{t-1} \quad (22)$$

Where r_t can be updated by:

$$r_t = r_{t-1} + \eta_{t-1} A d_{t-1} \quad (23)$$

and after several derivations of β_t [3], the simplified version of β_t can be obtained by:

$$\beta_t = \frac{r_t^{\mathrm{T}} r_t}{r_{t-1}^{\mathrm{T}} r_{t-1}} \quad (24)$$

The linear conjugate gradient algorithm is shown in **Algorithm Error!** No text of specified style in document.**.1**.

Algorithm Error! No text of specified style in document.**.1** Linear Conjugate Gradient Method (CG) [34]

Input: A, b, θ_0

Output: The solution θ^*

set $r_0 \leftarrow A\theta_0 - b$, $d_0 \leftarrow -r_0$, $t \leftarrow 0$;

while "Unsatisfied convergence condition" **do**

$$\eta_t \leftarrow \frac{r_t^{\mathrm{T}} r_t}{d_t^{\mathrm{T}} A d_t}$$

$$\theta_{t+1} \leftarrow \theta_t + \eta_t d_t$$

$$r_{t+1} \leftarrow r_t + \eta_t A d_t$$

$$\beta_{t+1} \leftarrow \frac{r_{t+1}^{\mathrm{T}} r_{t+1}}{r_t^{\mathrm{T}} r_t}$$

$$d_{t+1} \leftarrow -r_{t+1} + \beta_{t+1} d_t$$

$$t \leftarrow t + 1$$

end (while)

It was found that the CG method can be viewed as a minimization algorithm for the convex quadratic function.

Fletcher-Reeves (FR), and Polak-Ribiere (PR) methods have been introduced, as an extension of the conjugate gradient method, that are capable to optimize general convex and other non-linear functions [3]. Two simple changes have been made in **Algorithm Error!** No text of specified style in document..**1**. First, the step size η_t should perform a line search that identifies an approximate minimum of the non-linear function f along d_t. Second, the residual r, gradient of $F(\theta)$, must be replaced by the gradient of the non-linear objective function f. These modifications lead to the following non-linear optimizations **Algorithm Error!** No text of specified style in document..**2** Fletcher-Reeves Method (FR) and **Algorithm Error!** No text of specified style in document..**3** Polak-Ribiere Method (PR).

Algorithm Error! No text of specified style in document..**2** Fletcher-Reeves Method (FR) [3]

Input: A, b, θ_0

Output: The solution θ^*

set $f(\theta_0) = \|A\theta_0 - b\|^2$

set $r_0 \leftarrow \nabla f(\theta_0)$, $d_0 \leftarrow -r_0$, $t \leftarrow 0$;

while "Unsatisfied convergence condition" **do**

$update\ \{\eta_t, \theta_{t+1}, r_{t+1}, \beta_{t+1}, d_{t+1}\}$

$t \leftarrow t + 1$

end (while)

Algorithm Error! No text of specified style in document..**3** Polak-Ribiere Method (PR) [3]

Input: A, b, θ_0

Output: The solution θ^*

set $f(\theta_0) = \|A\theta_0 - b\|^2$

set $r_0 \leftarrow \nabla f(\theta_0)$, $d_0 \leftarrow -r_0$, $t \leftarrow 0$;

while "Unsatisfied convergence condition" **do**

$$\beta_{t+1}^{PR} \leftarrow \frac{r_{t+1}^{\mathrm{T}}(r_{t+1} - r_t)}{r_t^{\mathrm{T}} r_t}, \qquad \beta_{t+1} \leftarrow \beta_{t+1}^{PR}$$

$update\ \{\eta_t, \theta_{t+1}, r_{t+1}, \beta_{t+1}, d_{t+1}\}$

$t \leftarrow t + 1$

end (while)

The CG algorithm is graceful, that is, generating a new vector d_t depends only on the previous vector d_{t-1}, which does not require the knowledge or store all the prior vectors $d_0, d_1, d_2, \dots, d_{t-2}$. CG reduces the demands of high computational power as it is an $O(N)$ method.

Newton's method is designed to use both orders of derivatives, the gradient, which is also called the Jacobian matrix (first-order), and the Hessian matrix (second-order), to minimize the loss function with the quadratic function and to find the minimum value of the function. This process is iterated while a satisfying convergence occurs. Recall the main equation **Error! Reference source not found.** for Newton's method and generalize the formula for high dimension variable, that is:

$$\theta_{t+1} = \theta_t - \nabla^2 f(\theta_t)^{-1} \nabla f(\theta_t) \qquad (25)$$

where $\nabla^2 f$, ∇f are Hessian matrix and gradient of f respectively. The formula $\theta t+1= \theta_t - \eta \left(\frac{\partial^2 L(\theta)}{\partial \theta^2}\right)^{-1} \frac{\partial L(\theta)}{\partial \theta}$ (15) is derived from Taylor series expansion on $\theta_t = \theta_{t+1}$, that is:

$$f(\theta_t) \approx f(\theta_{t+1}) + \nabla f(\theta_{t+1})^{\mathrm{T}}(\theta_t - \theta_{t+1}) + \frac{1}{2}(\theta_t - \theta_{t+1})^{\mathrm{T}} \nabla^2 f(\theta_{t+1})(\theta_t - \theta_{t+1}) \qquad (26)$$

More exactly if step size is existed, the Newton's method formula can be written as shown:

$$\begin{cases} d_t = -\nabla^2 f(\theta_t)^{-1} \nabla f(\theta_t), \\ \theta_{t+1} = \theta_t + \eta_t d_t \end{cases} \qquad (27)$$

where d_t is the movement direction and η_t is the learning rate. Since the original Newton's method has computation and storage challenges, a Quasi-Newton method has been proposed. Quasi-Newton [35] uses Hessian's inverse estimate, $H_t = B_t^{-1}$, to calculate every iteration update by using some available estimations. Broyden-Fletcher-Golfarb-Shanno (BFGS) [3] is the most popular estimate algorithm. The inverse estimate of Hessian, $H_t^{(BFGS)}$ is calculated at every iteration as the following equations:

$$\begin{cases} B_{t+1}^{(BFGS)} = B_t - \frac{B_t s_t s_t^T B_t}{s_t^T B_t s_t} + \frac{u_t u_t^T}{u_t^T s_t} \\ H_{t+1}^{(BFGS)} = \left(I - \frac{s_t u_t^T}{s_t^T u_t}\right) H_t \left(I - \frac{u_t s_t^T}{s_t^T u_t}\right) + \frac{u_t s_t^T}{s_t^T u_t} \end{cases} \quad (28)$$

where, I is the identity matrix, $s_t = \theta_{t+1} - \theta_t$ and $u_t = \nabla f(\theta_{t+1}) - \nabla f(\theta_t) = B_{t+1} s_t$. The search direction of quasi-Newton method is:

$$d_t = -H_t g_t \quad (29)$$

where $g_t = \nabla f(\theta_t)$ is the gradient of f, and the update formula is:

$$\theta_{t+1} = \theta_t + \eta_t d_t \quad (30)$$

The BFGS **Algorithm Error!** No text of specified style in document.**.4** is shown below.

A novel method named Stochastic Variance Reduced Nesterov Accelerated Quasi-Newton method (SVR-NAQ) was introduced to solve the problem of high stochastic variance noise incurred when combining Nesterov and Quasi-Newton together [36], the high stochastic variance noise leads to slowing down the convergence.

Algorithm Error! No text of specified style in document.**.4** Broyden-Fletcher-Golfarb-Shanno (BFGS)

Input: $\theta_0 \in R^n, \epsilon > 0$

Output: The solution θ^*

set $g_0 \leftarrow \nabla f_0$, $u_0 \leftarrow 1$, $s_0 \leftarrow 1$, $H_0 \leftarrow \frac{s_t^T u_t}{\|u_t\|^2} I$, $t \leftarrow 0$;

while $\|g_t\| > \epsilon$ **do**

$d_t \leftarrow -H_t g_t$

$\theta_{t+1} \leftarrow \theta_t + \eta_t d_t$

where η_t is computed by using line searches $\min_{\eta_t} f(\theta_t + \eta_t d_t)$

$s_t \leftarrow \theta_{t+1} - \theta_t$

$u_t \leftarrow g_{t+1} - g_t$

update $H_{t+1}^{(BFGS)}$ from **Error! Reference source not found.**

$t \leftarrow t + 1$

end (while)

Quasi-Newton has a computation cost of $O(N^2)$ instead of $O(N^3)$ compared to original Newton's method. However, the storage needs of NxN matrices remains. A limited memory version of the quasi-Newton method called L-BFGS [37] [38] is designed to reduce the memory space required for every iteration.

2.1.3. Hessian-free method

Hessian-free [8] method, or truncated-Newton method, runs first by approximating a scaled-down copy of Hessian to locate the local curvature. Hessian-free uses conjugate gradient for optimization. While the original Hessian is too expensive to be computed in every iteration, the Hessian-free method employs a scaled-down copy of Hessian matrix, Hv with finite differences at the cost of a single extra gradient evaluation by identity:

$$Hv = \lim_{\varepsilon \to +0} \frac{\nabla f(\theta + \varepsilon v) - \nabla f(\theta)}{\varepsilon} \quad (31)$$

The approximate Hessian implemented by the Hessian-free algorithm is innovative since the matrix-vector products are needed only to optimize the quadratic objective functions. However, the method only works with conjugate gradient assistance. Conjugate gradient normally makes tremendous progress in optimizing for training iterations.

Algorithm Error! No text of specified style in document.**.5** Hessian Free Optimization Method [8]

Input: $\theta_0, \nabla f(\theta_0), \lambda$

Output: The solution θ^*

set $t \leftarrow 0$;

while "Unsatisfied convergence condition" **do**

$g_t \leftarrow \nabla f(\theta_t)$

Compute λ *by some methods*

$B_t(v) \equiv H(\theta_t)v + \lambda v$

Compute the step size η_t

$d_t \leftarrow CG(B_t, -g_t)$

$$\theta_{t+1} \leftarrow \theta_t + \eta_t d_t$$
$$t \leftarrow t + 1$$

end (while)

The Hessian Free Optimization **Algorithm Error!** No text of specified style in document.**.5** is shown above, where d_t is the search direction. HF method applies the CG method to compute an approximate solution d_t for the linear system.

$$B_t d_t = -\nabla f(\theta_t) \quad (32)$$

B_t is the Hessian matrix and is often defined as:

$$B_t = H(\theta_t) + \lambda I, \ \lambda \geq 0 \quad (33)$$

Hessian Free Optimization reduces the computation cost to $O(N^2)$ instead of $O(N^3)$ compared to original Newton's method. However, the storage needs of NxN matrices remain.

The following tables: Table 1 and Table 2, outline the First Order and second-Order optimizations methods consequently. Summarizing their features, pros, and cons.

Table 1 First-Order Optimization Methods Outline

Method	Features	Pros	Cons
GD	Solves the optimal value and converges at a linear rate. The method takes all training samples and compute the gradient at once.	When the objective function is convex, the solution is globally optimal.	The calculation cost is high as the gradients of total samples must be calculated in order to update every parameter.
SGD	Uses a random sample to update the model parameters in every iteration.	Faster than GD since it takes random samples for every iteration instead of the total number of samples.	Choosing a proper learning rate is difficult. The method can be trapped into saddle points in some cases.
NAG	Momentum method is used to accelerate the convergence speed.	The method is robust when dealing with unstable gradients which change the gradients' direction suddenly.	Choosing a proper learning rate is difficult.
AdaGrad	The learning rate is tuned based on all previous gradients over the time.	Gives better results in early iterations. Tunes the learning rate over the time.	As training iterations increases, the learning rate reaches a value zero. Choosing manual learning rates are mandatory.
AdaDelta / RMSProp	Updates the model parameter based on the accumulation of exponential moving average gradients.	Enhancement for AdaGrad at later stages. The method is suitable for optimizing non-convex and non-stationary problems.	The method can be trapped into saddle points in some cases during the late training phase.
Adam	Merges the adaptive methods with the momentum method.	The method is relatively stable for most optimization problems with large-scale data.	In some cases, the method does not converge.

Table 2 Second-Order Optimization Methods Outline

Method	Features	Pros	Cons
Newton's Method	The method retrieves more information from the inverse matrix of the Hessian matrix.	Provide a faster convergence speed compared to first-order methods since	The method requires heavy computation, and it makes the iteration step slow
Conjugate Gradient	It is a method of optimization between methods of the first and second order gradients.	The method needs to calculate the first order gradient only. The	The method is suitable for batch learning only. conjugate gradient computation is more complex

		method converges faster than the gradient descent method.	compared with the first-order gradient method.
Quasi-Newton Method	This method employs some estimation algorithms to get approximation of the Hessian matrix or its inverse matrix.	This method does not require heavy calculations compared to Newton's method.	It has to be run on a large storage space; consequently, it is not suitable for large-scale problems.
Hessian Free Method	The HF uses the CG to approximate the inverse of the Hessian matrix	The HFO employs the second-order gradient information from the estimates. It is consequently useful for high dimensional data.	The method is not designed to handle large-scale problems.

3. METHODOLOGY

In this thesis, the proposed method aims to reduce the number of iterations by implementing a second-order Newton's method to optimize the neural network, Hessian Free Optimization from Martens [8] and Martens and Sutskever [39]. This work also aims at investigating the integration between acceleration techniques with Hessian Free Optimization.

3.1. Hessian Free Optimization

As mentioned in Chapter2, the Hessian-free method works first by approximating a scaled-down copy of Hessian to define the local curvature and uses conjugate gradient (CG) for optimization. The strength of the CG process is another attractive feature of the HF approach. Unlike the non-linear CG approach (NCG) commonly used in machine learning, linear CG makes good use of the quadratic complexity of the optimization problem it solves in order to iteratively generate a set of "conjugate paths" and optimize them independently and precisely. In particular, the movement along every direction is precisely what Newton's method prefers; the reduction divided by the curvature that follows from the conjugation property. On the other hand, when applying the non-linear CG method, the directions it generates do not stay conjugated for a very long time, only approximately, and the line search is typically performed incorrectly and at a fairly high expense.

Hessian Free Optimization has been implemented as in [8], and [39], the same steps have been followed to make HFO work properly with for optimization of the neural network **Algorithm Error!** No text of specified style in document..**5**. HFO is also integrated with acceleration techniques such as Momentum (Nesterov Accelerated Gradient Descent) and Root Mean Square Propagation (RMSProp). Both techniques take the gradient (delta) from the conjugate gradient algorithm to accelerate the convergence based on previous steps.

HFO with momentum

The proposed method applies the momentum technique to Hessian free optimization on the Gradient (delta) derived from the conjugate gradient algorithm. Consequently, recalling the equations from Chapter **Error! Reference source not found.**, $v=\eta \cdot \left(-\frac{\partial L(\theta)}{\partial \theta}\right) + v^{old} \cdot mtm$ (8) and $\theta t+1= \theta_t + \eta_t d_t$ (20), the update on the CG algorithm by substitution both equations is:

$$\begin{cases} \Delta_{t+1}= \Delta_t + \eta_t d_t \\ v = \Delta_{t+1} + v^{old} \cdot mtm \\ \theta_{t+1} = \theta_t + \alpha v \end{cases} \quad (34)$$

where Δ is the CG update, v is the accumulated gradient and mtm is the momentum ranging between [0, 1].

Substituting **Error! Reference source not found.** on **Algorithm Error!** No text of specified style in document..1 Linear Conjugate Gradient Method (CG) produces the **Algorithm Error!** No text of specified style in document..**6**.

Algorithm Error! No text of specified style in document..**6** Linear Conjugate Gradient Method with Momentum

Input: A, b, θ_0

Output: The solution θ^*

set $r_0 \leftarrow A\theta_0 - b, d_0 \leftarrow -r_0, t \leftarrow 0;$

while "Unsatisfied convergence condition" **do**

$$\eta_t \leftarrow \frac{r_t^{\mathrm{T}} r_t}{d_t^{\mathrm{T}} A d_t}$$

$$\begin{cases} \Delta_{t+1} = \Delta_t + \eta_t d_t \\ v = \Delta_{t+1} + v^{old} \cdot mtm \\ \theta_{t+1} = \theta_t + \alpha v \end{cases}$$

$$r_{t+1} \leftarrow r_t + \eta_t A d_t$$

$$\beta_{t+1} \leftarrow \frac{r_{t+1}^{\mathrm{T}} r_{t+1}}{r_t^{\mathrm{T}} r_t}$$

$$d_{t+1} \leftarrow -r_{t+1} + \beta_{t+1} d_t$$

$$t \leftarrow t + 1$$

end (while)

HFO with RMSProp

Recall the equations from chapter2 $V_t = \sqrt{\beta V_{t-1} + (1-\beta)(g_t)^2}$ (11) and $\theta_{t+1} = \theta_t + \eta_t d_t$ (20)the update on the CG algorithm by substitution both equations is:

$$\begin{cases} \Delta_{t+1} = \Delta_t + \eta_t d_t \\ V_t = \sqrt{\beta V_{t-1} + (1-\beta)(\Delta_t)^2} \\ \theta_{t+1} = \frac{\alpha \Delta_t}{V_t + \epsilon} \end{cases} \quad (35)$$

where β is the parameter of exponential decay. Substituting $\begin{cases} \Delta_{t+1} = \Delta_t + \eta_t d_t \\ V_t = \sqrt{\beta V_{t-1} + (1-\beta)(\Delta_t)^2} \\ \theta_{t+1} = \frac{\alpha \Delta_t}{V_t + \epsilon} \end{cases}$ (35) on Algorithm Error! No text of specified style in document..1 Linear Conjugate Gradient Method (CG) produces the Algorithm Error! No text of specified style in document..**7**.

Algorithm Error! No text of specified style in document..**7** Linear Conjugate Gradient Method with RMSProp

Input: A, b, θ_0

Output: The solution θ^*

set $r_0 \leftarrow A\theta_0 - b, d_0 \leftarrow -r_0, t \leftarrow 0;$

while "Unsatisfied convergence condition" **do**

$$\eta_t \leftarrow \frac{r_t^{\mathrm{T}} r_t}{d_t^{\mathrm{T}} A d_t}$$

$$\begin{cases} \Delta_{t+1} = \Delta_t + \eta_t d_t \\ V_t = \sqrt{\beta V_{t-1} + (1-\beta)(\Delta_t)^2} \\ \theta_{t+1} = \frac{\alpha \Delta_t}{V_t + \epsilon} \end{cases}$$

$$r_{t+1} \leftarrow r_t + \eta_t A d_t$$

$$\beta_{t+1} \leftarrow \frac{r_{t+1}^{\mathrm{T}} r_{t+1}}{r_t^{\mathrm{T}} r_t}$$

$$d_{t+1} \leftarrow -r_{t+1} + \beta_{t+1} d_t$$

$$t \leftarrow t + 1$$

end (while)

3.2. Hybrid Hessian Free Optimization

This section explains the hybrid algorithm for combining two-degree orders, first-order, and second-order optimization. The hybrid algorithm uses the first-order method for a set number of steps followed by the second-order method, which decreases the calculation time relative to the second-order method alone, making the convergence much faster than the first-order method alone. This is an in-between method.

A set number of parameters have been tested to evaluate this method, but it was found that for every dataset, there was an optimal number; consequently, an automatic algorithm has been proposed to switch automatically between the first-order and second-order methods.

Algorithm Error! No text of specified style in document..**8** Hybrid Optimization Algorithm (HOA)

Input: θ

Output: The solution θ^*

set $Lr \leftarrow 0.1, S_0 \leftarrow 0, S_1 \leftarrow 15;$

while "Unsatisfied convergence condition" **do**

$$compute\ training\ loss, L_{t+1}$$

$$r \leftarrow \frac{L_{t+1}}{L_t}$$

If $r < 1 + Lr$ **or** $r > 1 - Lr$ **then**

$$S_1 = \frac{S_1}{r}$$

end (if)

If $S_0 \geq S_1$ **then**

$S_0 = 0$

run_second()

else

run_first()

end (if)

$S_0 = S_0 + 1$

end (while)

The **Algorithm Error!** No text of specified style in document.**.8** shows the step for the hybrid method, where Lr is the loss residual range, r is the loss residual, S_0 is the current step number and S_1 is the step number ceiling to execute the second-order method.

Since the hybrid algorithm uses Hessian Free Optimization and Gradient Descent Optimization, the computation cost for the method is $O(N^2)$ and the storage needs of NxN matrices is needed as well.

4. EXPERIMENTS

In this section, all experiments, including datasets description, model architecture, training process, and results, are covered.

4.1. Numerical Example

In this section, a numerical example has been applied for most of the optimization methods to minimize the value of a two-independent variable function. The example uses Newton's method, gradient descent method, conjugate gradient descent, quasi-Newton method, and Hessian free optimization method this example explain and justify the results analysis criteria by means of which a conclusion was reached.

Let the function $f(x_1, x_2) = x_1^2 - 2 \cdot x_1 x_2 + 4 \cdot x_2^2$ consequently the Hessian matrix of $f(x_1, x_2)$, $H = \begin{bmatrix} 2 & -2 \\ -2 & 8 \end{bmatrix}$, figure 1 illustrate the contour lines for the given example and figure 2 give the surface visilization for the same function. The initial values of the parameters $(x_1, x_2) = (-3,2)$ and the global minimum occurs when the parameter have the value of $(0,0)$. The optimizers have been applied until they reached satisfied convergence table 3 gives the updated parameters for all the iterations.

4.2. Datasets Used

For experiments, 6 different data sets have been considered, as shown in Table 4. Where N_0 is the number of features, l is the number of training instances, l_t is the number of testing instances and K is the number of classes. All datasets, come with training sets and testing sets expect for Sensorless is split.

Table 3 The Updated Parameters (x_1,x_2) for every iteration

Iteration	(x_1, x_2) Values				
	Newton's Method	GD	CG	QN	HFO
1	(-3.00, 2.00)	(-3.00, 2.00)	(-3.00, 2.00)	(-3.00, 2.00)	(-3.00, 2.00)
2	(0.00, 0.00)	(-1.50, -1.30)	(-1.50, -1.30)	(-2.00, -0.20)	(-1.01, -0.46)
3	-	(-1.44, -0.19)	(-1.30, -0.50)	(-1.46, -0.13)	(-0.19, 0.12)
4	-	(-1.06, -0.39)	(-1.04, -0.22)	(-1.00, -0.19)	(0.02, 0.01)
5	-	(-0.86, -0.24)	(-0.64, -0.10)	(-0.73, -0.26)	(0.02, 0.00)
6	-	(-0.68, -0.21)	(-0.27, -0.11)	(-0.44, -0.20)	(0.00, 0.00)
7	-	(-0.54, -0.16)	(-0.17, -0.06)	(-0.12, -0.06)	-
8	-	(-0.42, -0.13)	(-0.10, -0.02)	(-0.03, 0.01)	-
9	-	(-0.34, -0.10)	(-0.05, -0.01)	(0.01, -0.00)	-

4.3. Networks Architecture

For all experiments, the following neural network architecture that has been implemented is explained below for all datasets. The four measurements (training loss, training accuracy, testing loss, and testing accuracy) have been compared between the first-order technique, gradient descent, and second-order optimization, the Hessian Free Optimization.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The neural network architecture consists of the input shape, and it depends on the dataset used, followed by two hidden layers with 200 and 100 hidden nodes respectively, transformed to a non-linear form using sigmoid activation function, and finally, a SoftMax layer and the layer's output shape depends also on the dataset that would be classified as shown in Figure 3. For the sake of simplicity, all experiments were done on the same network architecture, but with different learning, rates to figure out what is the best learning rate used for every optimization algorithm and for every dataset.

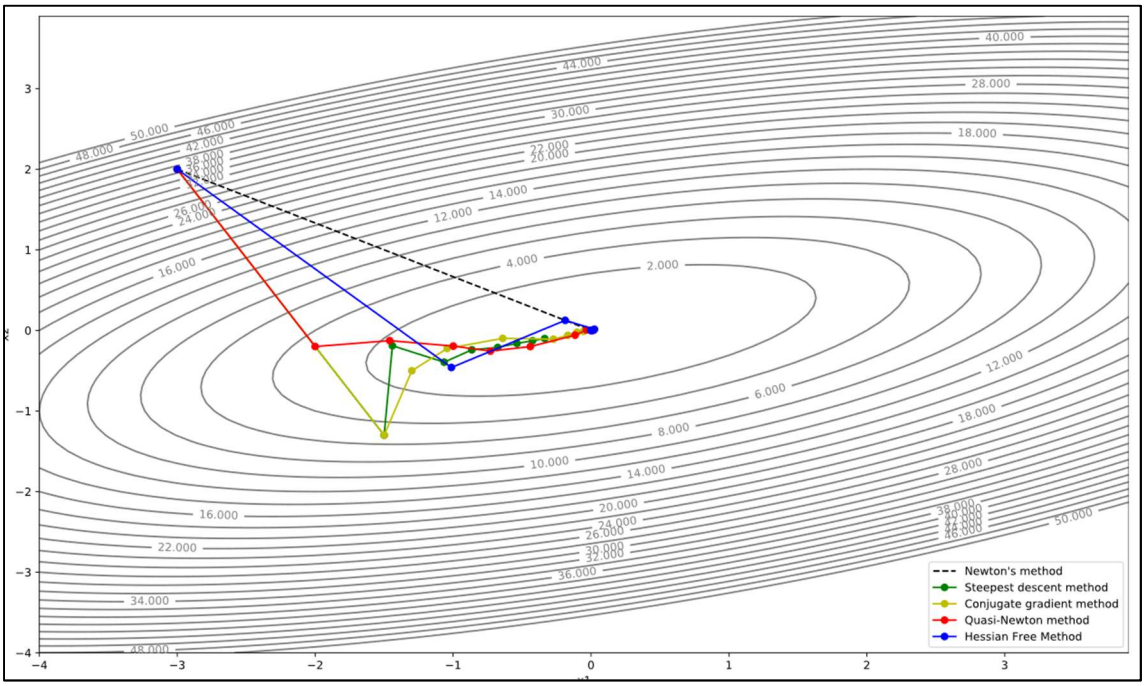

Figure 1 Contour Lines for $x_1^2 - 2.x_1x_2 + 4x_2^2$ and optimizations

Journal of Theoretical and Applied Information Technology
30th November 2021. Vol.99. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

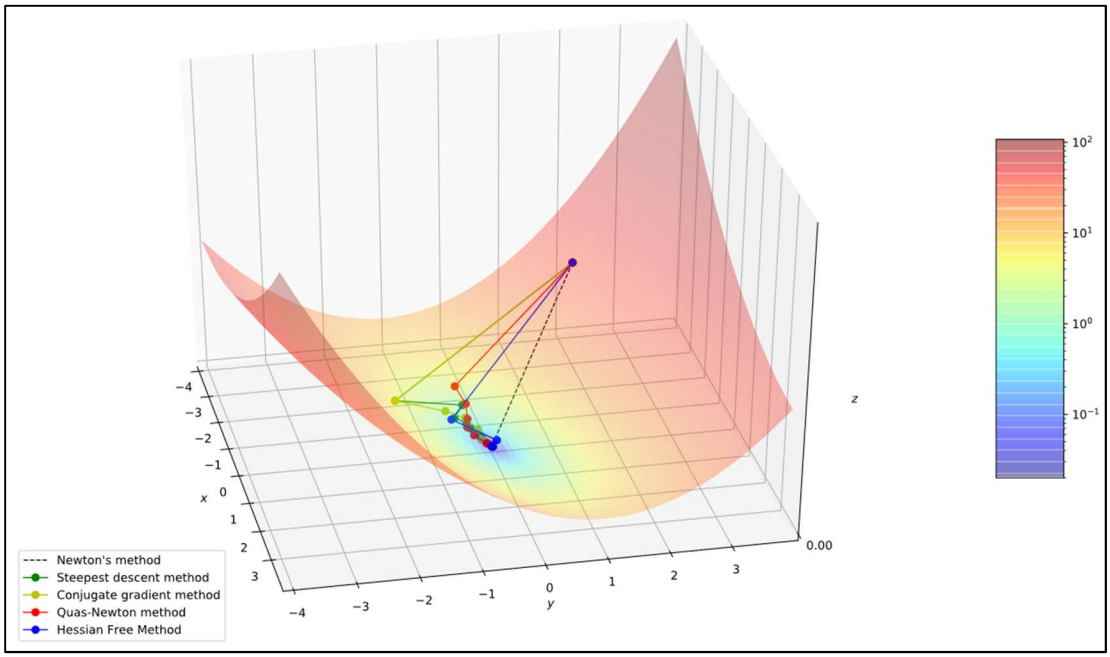

Figure 2 Surface of $x_1^2 - 2.x_1 x_2 + 4.x_2^2$ and optimizations

Table 4 Datasets Summary.

Dataset	Dataset Description	N_0	Scaled Values	l	l_t	K
Letter	This collection is from the Statlog collection [40] and values have been scaled to [−1, 1] for every of the features.	16	(-1.0, 1.0)	15,000	5,000	26
MNIST	This handwritten digit recognition data set [41] is commonly used for benchmarking classification algorithms. A scaled edition has been considered, in which 255 divides every value of the features.	784	(0.0, 1.0)	60,000	10,000	10
Pendigits	This set of data originated in Alimoglu and Alpaydin [42].	16	(0, 100)	7,494	3,498	10
Satimage	This set is from the Statlog collection [40] and values have been scaled in [−1, 1] for every of the features.	36	(-1.0, 1.0)	4,435	2,000	6
SensIT	Duarte and Hu [43] data collection contains signals from acoustic and seismic sensors to identify the different vehicles. The original edition has been used, unscaled edition.	100	(-0.811, 1.0)	78,823	19,705	3
Sensorless	The Paschke et al. [44] data collection. Values have been scaled for every function to be in [0, 1], and then conducted stratified random sampling to pick 10,000 instances to be the test set and the rest of the data to be the training set.	48	(0.0, 1.0)	48,509	10,000	11

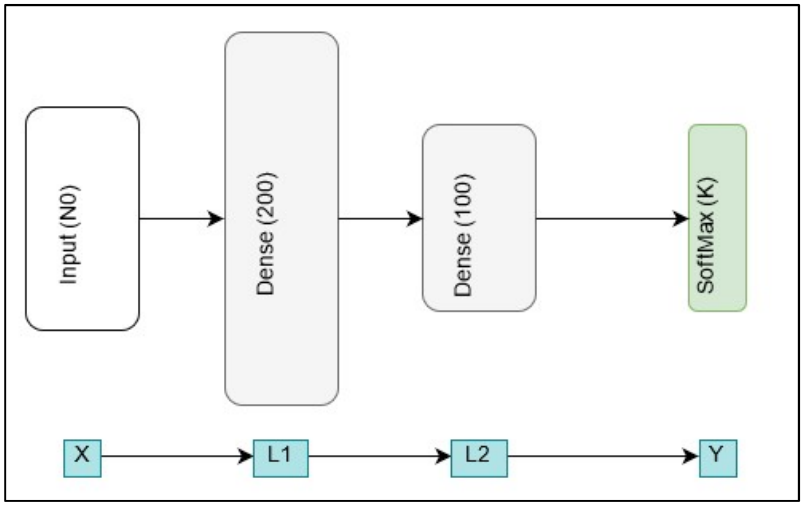

Figure 3 Neural Network Architecture

Below, Table 5, illustrates the done experiments in detail for every dataset. The number of done experiments is 486, 81 for every dataset. In the next section 4.4 below, we compared the results among the optimizers.

Table 5 Experiments Summary

Dataset	Optimization Type	Acceleration and Learning Rates
Letter	GD HFO Hybrid_auto (1epoch, 20step, 30step, 1,2,3,4)	Normal: [0.25, 0.30, 0.35] Momentum: [0.005, 0.006, 0.007] RMSProp: [0.0005, 0.0006, 0.0007]
MNIST		
Pendigits		
Satimage		
SensIT		
Sensorless		

4.4. Experimental Part

In this section, a comparison of different optimizers on different datasets is shown. All experiments were done on Google Cloud Virtual Machine (VM), using TensorFlow v1.5 running on a single CPU core.

The datasets were selected because they were available online, and most of the studies were based on these datasets [45], [46]. A single unified neural network architecture has been implemented for all datasets because the cloud VM used for this thesis is not big enough to manage extremely complex computation. The parameters of the neural network architecture have been optimized using the first-order methods, second-order methods, and hybrid methods.

The number of conducted experiments was 486 (6 datasets x 9 optimization methods x 3 acceleration techniques x 3 different learning rates). Details are shown in Table 5. For the sake of simplicity, only the best model of every distinguished optimization technique has been selected. The distinguished techniques are shown in Table 6.

Every model was trained on the same neural network architecture for the same batch size (100), and the training was terminated after reaching 3000 steps. Consequently, optimization techniques were the only variable in all experiments for every dataset. Figure 5 shows the best testing loss, for each optimization order, for every dataset.

The outcomes of the techniques are summarized in Table 8. The second-order optimization method with RMSprop acceleration surpassed all other techniques in four datasets out of six in terms of achieving the lowest training loss. For the other two datasets, the hybrid technique with RMSprop achieved the lowest training loss. In general, RMSprop was better than the momentum acceleration technique and better than the optimization method without an acceleration in terms of accuracy.

Table 6 Distinguished Optimization Techniques

Order-Degree	Optimization
First-Order	Gradient Descent
	Gradient Descent with Momentum
	Gradient Descent with RMSProp
Second-Order	Hessian Free Optimization
	Hessian Free Optimization with Momentum
	Hessian Free Optimization with RMSProp
Hybrid	Hybrid: Gradient Descent and Hessian Free Optimization
	Hybrid with Momentum
	Hybrid with RMSProp

For the Letter dataset and Sensorless dataset, second-order optimization outperformed both the first-order and the hybrid techniques significantly. Second, with RMSprop achieved a training loss of 0.73 while the next method was limited to 0.95 on the Letter dataset. Same for the sensorless dataset with 0.32 for the second-order, followed by 0.89 for achieved by the first order. The second-order and the hybrid achieved similar results on MNIST and Satimage datasets with a slight advantage for the second order. The difference between the training loss was around 0.2 as the results were 0.11 for the second order and 0.13 for the hybrid on MNIST, while the training loss on Satimage was 0.31 for the 2nd order and 0.33 for the hybrid. However, the hybrid surpassed the second-order significantly on PenDigits with 0.1 training loss for the hybrid against 0.4 for the second order.

The reason why the hybrid technique has achieved better results in two datasets (PenDigits and SensIT) is due to the nature of the datasets, especially in the PenDigits dataset, explained in Figure 5. PenDigits has achieved a very smaller testing loss compared to the second-order methods.

Although the hybrid technique with RMSprop did not surpass the second-order technique with RMSprop in most datasets in terms of testing loss, the hybrid technique was better than the first order in five datasets out of six.

PenDigits has a relatively small input dimension with a broader range in the input values (0-100), which cannot be scaled due to the nature of the dataset. This contributes to the presence of high stochastic variance noise problems, which make the convergence more slowly and often unstable [36]. Figure 4 shows the instability of the PenDigits testing loss.

The hybrid method was selected properly among four hybrid parameters to figure out the best parameters for this method. Table 9 shows that the hybrid method with AUTO_3 parameters integrated with RMSProp has the best testing loss in four datasets out of six. For the remaining two datasets, the difference was minimal (less than 3.5%). Table 7 shows the parameters used in every hybrid method.

Table 7 Hybrid Optimization Method Params

Methods/Params	Lr	S_0	S_1
AUTO_1	0.1	0	20
AUTO_2	0.1	0	17
AUTO_3	0.1	0	10
AUTO_4	0.1	0	15

Journal of Theoretical and Applied Information Technology
30th November 2021. Vol.99. No 22

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

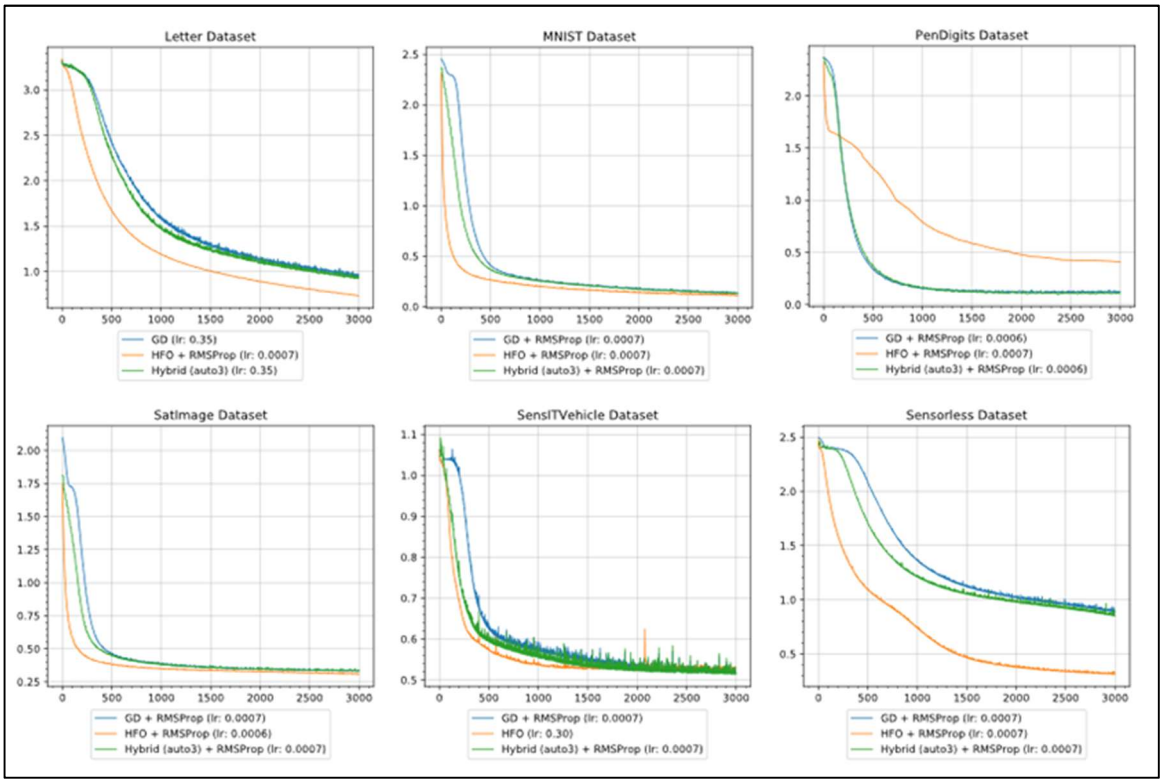

Figure 4 Testing Loss Plots for every Dataset (Best Optimizers)

Table 8 Results Comparison (Testing Loss)

Optimization/ Dataset		Letter	MNIST	PenDigits	Satimage	SensIT	Sensorless
1st order	GD	0.9614	0.1976	0.1497	0.3504	0.5399	1.0498
	Momentum	2.1800	0.3762	0.1726	0.4295	0.6097	1.6149
	RMSProp	1.0028	0.1339	0.1240	0.3324	0.5171	0.8897
2nd order	HFO	0.7668	0.1734	0.9915	0.3338	0.5277	0.5943
	Momentum	1.5682	0.2948	0.8061	0.3910	0.5450	1.1129
	RMSProp	0.7342	0.1084	0.4077	0.3108	0.5469	0.3155
Hybrid	Hybrid	0.9351	0.1951	0.3094	0.3493	0.5384	1.3768
	Momentum	2.0081	0.3814	0.1775	0.4259	0.6053	1.5429
	RMSProp	0.9728	0.1269	0.1045	0.3341	0.5162	0.8534

Table 9 Results Comparison between Hybrid Methods (Testing Loss)

Optimization/ Dataset	Letter	MNIST	PenDigits	Satimage	SensIT	Sensorless
AUTO_1	0.9491	0.1929	0.1430	0.3490	0.5433	0.9726
AUTO_1 + Momentum	2.0603	0.3728	0.1676	0.4294	0.6092	1.5982

AUTO_1 + RMSProp	0.9928	0.1323	0.1121	0.3325	0.5167	0.8239
AUTO_2	0.9411	0.1932	0.1756	0.3503	0.5393	1.0452
AUTO_2 + Momentum	2.0597	0.3756	0.1701	0.4215	0.6096	1.5975
AUTO_2 + RMSProp	0.9675	0.1344	0.1248	0.3330	0.5167	0.8572
AUTO_3	0.9351	0.1951	0.3094	0.3493	0.5384	1.3768
AUTO_3 + Momentum	2.0081	0.3814	0.1775	0.4259	0.6053	1.5429
AUTO_3 + RMSProp	0.9728	0.1269	0.1045	0.3341	0.5162	0.8534
AUTO_4	0.9424	0.1947	0.3737	0.3525	0.5453	0.9232
AUTO_4 + Momentum	2.0423	0.3780	0.1798	0.4266	0.6054	1.5625
AUTO_4 + RMSProp	0.9721	0.1332	0.1268	0.3330	0.5174	0.8749

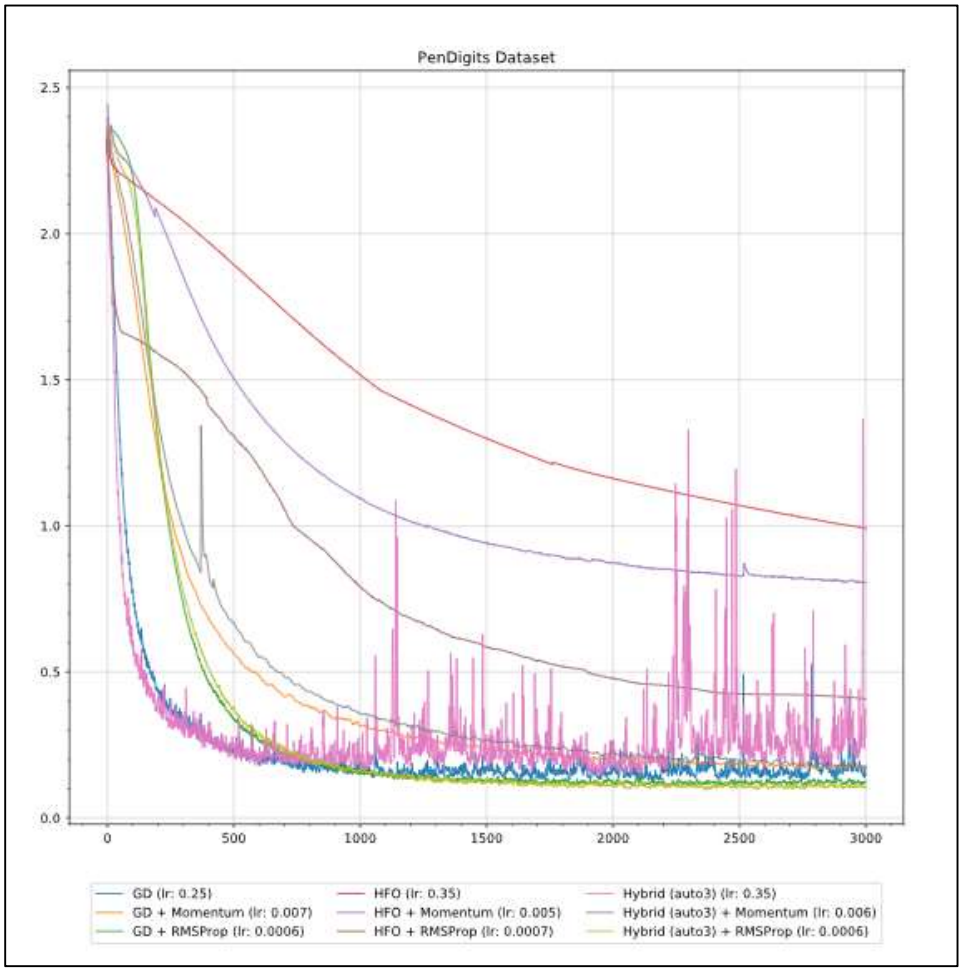

Figure 5 Testing Loss Plots for PenDigits Dataset

The hybrid technique was introduced to balance the model's performance and training time. Experiments have shown that the running time of the hybrid is almost equivalent to the first order. However, it is extremely better than the second order. Quantitatively, the running time of the second-order methods is 5-6 times slower than the hybrid, as shown in Figure 6.

Moreover, the number of steps and training time comparison for reaching set values of testing

loss was expressed, as shown in Table 10 and Table 11, respectively.

As a result, the hybrid method with the RMSProp acceleration technique is recommended for optimizing deep learning models when high performance is required in a short duration.

In Figure 6, Table 10, and Table 11, each experiment has been repeated 50 times to eliminate the possibility of the randomness in the reported results, and the average value has been reported. Moreover, the significance of the results was tested. Scientific Python library (Scipy) was used to conduct the hypothesis test by comparing the distribution of the results obtained from different models (i.e., the hybrid algorithm and the first-order algorithm). The p-values for all the tests were less than 0.001, which means that the reported results are scientifically significant.

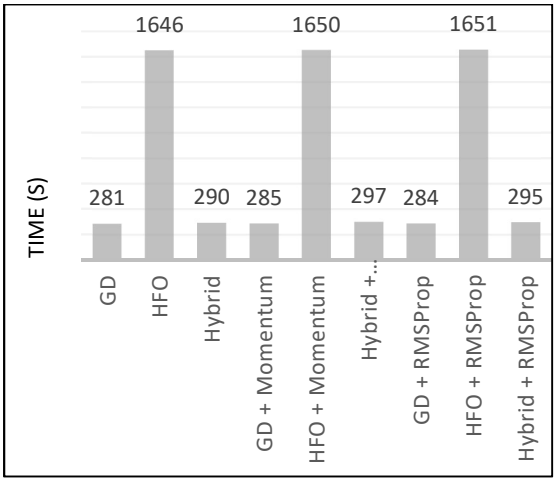

Figure 6 Time Comparison for Optimization Methods

Table 10 Number of Steps vs Testing Loss Reached

Optimization	GD	HFO	Hybrid	Testing Loss ≤
Steps Reached	285	37	196	1.00
	429	124	357	0.50
	1647	972	1538	0.20
	2604	1704	2417	0.15

Table 11 Training Time (Seconds) vs Testing Loss Reached

Optimization	GD	HFO	Hybrid	Testing Loss ≤
Training Time (s)	27	20	19	1.00
	41	68	35	0.50
	156	535	151	0.20
	247	938	238	0.15

5. CONCLUSION

This paper investigated the second-order methods of optimization and showed that utilizing the second derivative results in faster convergence by implementing and evaluating Newton's second-order optimization method and Hessian Free Optimization (HFO) on fully connected feed-forward networks. This thesis has proposed enhancing second-order methods by the integration with some acceleration techniques such as Momentum and Root Mean Square Propagation (RMSProp). The experimental results demonstrated the superiority of second-order methods in terms of performance, up to 28% enhancement on testing loss in four datasets out of six, despite the additional overhead (time and computation).

This paper proposed a novel hybrid algorithm capable of combining first-order and second-order optimization methods to balance the pros and cons of increasing the order of optimization methods. Four possible approaches to hybrid algorithms were developed and evaluated. The experimental results showed the superiority of the hybrid method. The experimental results showed that the proposed hybrid algorithm had balanced the trade-of between training time from one side and the performance of optimized models from the other side. The hybrid algorithms yielded better-optimized models than first-order methods, up to 5% enhancement on testing loss in five datasets out of six, while consuming almost the same training time. Moreover, the training time of the hybrid model was 5-6 times faster than second-order methods.

This research improved the efficiency of training deep learning models by decreasing the number of required iterations to achieve the desired

output of the models. This work provides the integration between the acceleration techniques with second-order Newton's method and reduced the number of iterations by implementing a second-order Newton's method to optimize the neural network.

For future work a hybrid model of three degrees of order will be investigated in addition to extending the evaluation to include deeper and more complex neural networks architectures.

REFERENCES:

[1] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[2] M. A. Otair and W. A. Salameh, "Speeding Up Back-Propagation Neural Networks," *Proceedings of the 2005 Informing Science and IT Education Joint Conference,* vol. 1, pp. 167-173, 2005.

[3] J. Nocedal and S. J. Wright, Numerical Optimization, Springer, 2006.

[4] M. M. Kasar, D. Bhattacharyya and T.-h. Kim, "Face Recognition Using Neural Network: A Review," *International Journal of Security and Its Applications,* vol. 10, pp. 81-100, 2016.

[5] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh and K. Shaalan, "Speech Recognition Using Deep Neural Networks: A Systematic Review," *IEEE Access,* vol. 7, pp. 19143-19165, 2019.

[6] Y. Goldberg, "A Primer on Neural Network Models for Natural Language Processing," *Journal of Artificial Intelligence Research* , vol. 57, pp. 45-420, 2016.

[7] J. E. Dennis, Jr. and J. J. Moré, "Quasi-Newton Methods, Motivation and Theory," *SIAM Review,* vol. 19, pp. 46-89, 1977.

[8] J. Martens, "Deep learning via Hessian-free optimization," in *International Conference on Machine Learning*, 2010.

[9] S. Sun, Z. Cao, H. Zhu and J. Zhao, "A Survey of Optimization Methods From a Machine Learning Perspective," *IEEE Transactions on Cybernetics,* pp. 1-14, 2019.

[10] J. Alspector, R. Meir, B. Yuhas, A. Jayakumar and D. Lippe, "A Parallel Gradient Descent Method for Learning in Analog VLSI Neural Networks," in *Advances in Neural Information Processing Systems 5*, Morgan-Kaufmann, 1993, pp. 836-844.

[11] R. Johnson and T. Zhang, "Accelerating Stochastic Gradient Descent using Predictive Variance Reduction," in *Advances in Neural Information Processing Systems*, 2013.

[12] A. S. Nemirovsky and D. B. Yudin, "Problem Complexity and Method Efficiency in Optimization," *SIAM Review,* vol. 27, no. 2, pp. 264-265, 1985.

[13] A. Nemirovski, A. Juditsky, G. Lan and A. Shapiro, "Robust Stochastic Approximation Approach to Stochastic Programming," *SIAM Journal on Optimization,* vol. 19, no. 4, pp. 1574-1609, 2009.

[14] A. Agarwal, P. L. Bartlett, P. Ravikumar and M. J. Wainwright, "Information-Theoretic Lower Bounds on the Oracle Complexity of Stochastic Convex Optimization," *IEEE Transactions on Information Theory,* vol. 58, no. 5, pp. 3235-3249, 2012.

[15] N. L. Roux, M. Schmidt and F. R. Bach, "A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets," in *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., 2012, pp. 2663-2671.

[16] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv e-prints,* p. arXiv:1609.04747, 2016.

[17] H. Robbins and S. Monro, "A Stochastic Approximation Method," *The Annals of Mathematical Statistics,* vol. 22, no. 3, pp. 400-407, 1951.

[18] C. Darken, J. Chang and J. Moody, "Learning Rate Schedules for Faster Stochastic Gradient Search," in *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, 1992, pp. 3-12.

[19] J. Duchi, E. Hazan and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research,* vol. 12, pp. 2121-2159, 2011.

[20] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations*, 2015, pp. 1-15.

[21] I. Sutskever, "Training Recurrent Neural Networks," Ph.D. dissertation, University of Toronto, Ontario, Canada, 2013.

[22] Z. Allen-Zhu, "Natasha 2: Faster Non-Convex Optimization Than SGD," in *Advances in Neural Information Processing Systems 31*, Curran Associates, Inc., 2018, pp. 2675-2686.

[23] R. Ge, F. Huang, C. Jin and Y. Yuan, "Escaping From Saddle Points --- Online Stochastic Gradient for Tensor Decomposition," in *Proceedings of The 28th Conference on Learning Theory*, PMLR, 2015, pp. 797-842.

[24] B. 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.

[25] F. Klinker, "Exponential moving average versus moving exponential average," *Mathematische Semesterberichte,* vol. 58, no. 1, pp. 97-107, 2011.

[26] I. Sutskever, J. Martens, G. Dahl and G. Hinton, "On the Importance of Initialization and Momentum in Deep Learning," in *International Conference on Machine Learning*, 2013.

[27] L. C. Baird III and A. W. Moore, "Gradient Descent for General Reinforcement Learning," in *Advances in Neural Information Processing Systems 11*, MIT Press, 1999, pp. 968-974.

[28] C. Darken and J. E. Moody, "Note on Learning Rate Schedules for Stochastic Optimization," in *Advances in Neural Information Processing Systems 3*, Morgan-Kaufmann, 1991, pp. 832-838.

[29] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *arXiv e-prints,* p. arXiv:1212.5701, 2012.

[30] T. Tieleman and G. Hinton, "Divide the Gradient by a Running Average of Its Recent Magnitude," *COURSERA: Neural Networks for Machine Learning,* vol. 4, pp. 26-31, 2012.

[31] H. H. Tan and K. H. Lim, "Review of Second-Order Optimization Techniques in Artificial Neural Networks Backpropagation," *IOP Conference Series: Materials Science and Engineering,* vol. 495, p. 012003, 2019.

[32] B. S. Goh, "Greatest Descent Algorithms in Unconstrained Optimization," *Journal of Optimization Theory and Applications,* vol. 142, no. 2, p. 275–289, 2009.

[33] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards,* vol. 49, no. 6, 1952.

[34] J. R. Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," Carnegie Mellon University, 1994.

[35] J. Sohl-Dickstein, B. Poole and S. Ganguli, "Fast Large-Scale Optimization by Unifying Stochastic Gradient and Quasi-Newton Methods," *arXiv e-prints,* p. arXiv:1311.2115, 2013.

[36] S. Yasuda, S. Mahboubi, S. Indrapriyadarsini, H. Ninomiya and H. Asai, "A Stochastic Variance Reduced Nesterov's Accelerated Quasi-Newton Method," *arXiv e-prints,* p. arXiv:1910.07939, 2019.

[37] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming,* vol. 45, no. 1, pp. 503-528, 1989.

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

[39] J. Martens and I. Sutskever, "Training Deep and Recurrent Networks with Hessian-Free Optimization," in *Neural Networks: Tricks of the Trade: Second Edition*, Springer Berlin Heidelberg, 2012, pp. 479-535.

[40] D. Michie, D. J. Spiegelhalter, C. C. Taylor and J. A. Campbell, Machine learning, neural and statistical classification, NJ, United States: Ellis Horwood, Imprint of Simon and Schuster One Lake Street Upper Saddle River, 1995.

[41] 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.

[42] F. Alimoglu and E. Alpaydin, "Methods of Combining Multiple Classifiers Based on Different Representations for Pen-based Handwritten Digit Recognition," in *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks*, 1996.

[43] M. F. Duarte and Y. H. Hu, "Vehicle classification in distributed sensor networks," *Journal of Parallel and Distributed Computing,* vol. 64, no. 7, pp. 826-838, 2004.

[44] F. Paschke, C. Bayer, M. Bator, U. Mönks, A. Dicks, O. Enge-Rosenblatt and V. Lohweg,

"Sensorlose Zustandsüberwachung an Synchronmotoren," in *Workshop Computational Intelligence (VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik (GMA))*, Pullman Hotel, Dortmund, Germany, 2013.

[45] C.-C. Wang, K. L. Tan, C.-T. Chen, Y.-H. Lin, S. S. Keerthi, D. Mahajan, S. Sundararajan and C.-J. Lin, "Distributed Newton Methods for Deep Neural Networks," *arXiv e-prints,* p. arXiv:1802.00130, 2018.

[46] C.-C. Wang, K. L. Tan and C.-J. Lin, "Newton Methods for Convolutional Neural Networks," *arXiv e-prints,* p. arXiv:1811.06100, 2018.

[47] M. A. Otair and W. A. Salameh, "Solving the Exclusive-OR Problem Using an Optical Backpropagation Algorithm," *2nd Jordanian International Conference on Computer Science and Engineering (JICCSE2006),* pp. 298-301, 2006.

[48] M. A. Otair and W. A. Salameh, "Online Handwritten Character Recognition Using an Optical Backpropagation Neural Networks," *In Proceedings of The 2004 International Research Conference on Innovations in Information Technology,* pp. 334-341.

[49] O. M. Surakhi and W. A. Salameh, "Enhancing the Performance of the BackPropagation for Deep Neural Network," *INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY,* vol. 13, no. 12, pp. 5274-5285, 2014.