Assignment 2 Option 2
Comparative Analysis of Stochastic Gradient Descent, Scaled Conjugate Gradient, and Leapfrog Optimisation Algorithms for Feed Forward Neural Networks

A.D. van der Merwe
Department of Computer Science
University of Stellenbosch
24923273
24923273@sun.ac.za

***Abstract*—In this report, the application of optimisation algorithms is explored, with a focus on the stochastic gradient descent (SGD) optimisation algorithm, the scaled conjugate cradient (SCG) optimisation algorithm, and the leap-frog optimisation (LFOP) algorithm. All three optimisation algorithms are implemented in a feed forward neural network (FFNN) model to compare these optimisation algorithms performances against one another. Classification tasks and function approximation tasks that vary in complexity are explored to thoroughly to compare the SGD optimisation algorithm, the SCG optimisation algorithm, and the LFOP algorithm with one another.**

I. Introduction

The optimisation algorithm used to adjust the weights of a feed forward neural network (FFNN) model significantly influences the performance of the model. Different optimisation techniques can impact convergence rates, performance, and general effectiveness when patterns from the data are learned. Optimisation algorithms have various strenghts and weaknesses, and to understand each of these strenghts and weaknesses is crucial for the selection of the most suitable approach for the specific task.

This paper focuses on three optimisation algorithms, namely the stochastic gradient descent (SGD) optimisation algorithm, the scaled conjugate cradient (SCG) optimisation algorithm, and the leap-frog optimisation (LFOP) algorithm. This paper conducts a comparative analysis of the SGD, SCG, and LFOP algorithms by the construction of a one hidden layer FFNN model that utilises each optimisation algorithm. The results show that the SGD optimisation algorithm performs exceptionally well in function approximation tasks and large datasets with complex data patterns. However the SGD optimisation algorithm is computationally expensive. The SCG algorithm quickly converges and works best with smaller datasets, while the LFOP algorithm outperforms the other optimisation algorithms on datasets that is more susceptible to a local minima.

To better understand these strengths and weaknesses, this paper implements three classification tasks and three function approximation tasks that vary in complexity.

The rest of this paper is structured as follows: Section II provides background information on the FFNN model, the SGD optimisation algorithm, the SCG optimisation algorithm, and the LFOP algorithm. Additionally, background on different activation functions, objective functions, performance metrics, and weight decay. Section III provides the implementation of the FFNN model, and the three optimisation algorithms. Section IV provides the empirical procedure and Section V provides the research results.

II. Background

This section presents background information on the FFNN model. Additionally, information on different activation functions and three different optimisation algorithms used to train the FFNN model. This section also presents background information on the different objective functions and performance metrics used to evaluate the FFNN model, as well as background on weight decay regularisation.

A. Feed Forward Neural Network

The concept of a FFNN was first introduced by Warren McCulloch and Walter Pitts in 1943 [9]. McCulloch and Pitts developed a basic binary model of a neuron that formed the basic idea of a FFNN. The model uses weighted inputs and a threshold to perform simple logical operations such as AND, OR, and NOT.

In 1958, Frank Rosenblatt developed one of the earliest practical implementations of a FFNN neural network called a Perceptron [12]. This single-layer network was designed to learn from data over time by the adjustment of the weights based on the input data to perform binary classification tasks. One drawback of the perceptron is that it solves only linearly separable problems.

In 1986, the backpropagation algorithm was developed by David Rumelhart, Geoffrey Hinton, and Ronald J. Williams [6]. The backpropagation algorithm enabled the ability to train multilayer FFNN that addressed the problems of non-linearly separable classes.

A FFNN consists of three layers, namely the input layer, hidden layer and output layer. A FFNN can have more than one hidden layer and only one input and output layer. To train a FFNN an iterative process with multiple passes through the training set is used. One pass through the training set is known as an epoch. There are two optimisation methods to use when a FFNN is trained, namely stochastic learning and batch learning.

When stochastic learning is applied, the weights are adjusted after each pattern that is presented. The patterns are randomly selected in each iteration to prevent the model from overfitting to the specific order of patterns in the training set, which helps the model generalise better. In batch learning, weight updates are gathered and used to adjust weights only after all training patterns have been presented.

The equation used to calculate the output neuron of a FFNN with one hidden layer is as follows:

$$o_{k,p} = f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I+1} v_{ji} z_{i,p}\right)\right) \tag{1}$$

where $o_{k,p}$ is the output of the k-th neuron for the p-th input pattern, y_j is the j-th neuron in the hidden layer, f_{o_k} and f_{y_j} are the activation function for output neuron o_k and hidden neuron y_j respectively, w_{kj} is the weight between output neuron o_k and hidden neuron y_j, v_{ji} is the weight between hidden neuron y_j and input neuron z_i, $z_{i,p}$ is the value of the i-th input neuron for the p-th input pattern and the $(I+1)$-th input neuron and the $(J+1)$-th hidden neuron are bias neurons that represents the threshold values of neurons in the next layer.

There are several optimisation algorithms that can be used when a FFNN is trained. These Algorithms are grouped into two classes, namely:

- Local optimisation, where the algorithm might never find the global minimum, as the algorithm might get stuck in a local minimum. Examples of local optimisation algorithms are SGD and SCG.
- Global optimisation, where the algorithm employes mechanisms to search larger parts of the search space to find the local minimum. An example of a global optimisation algorithm is the LFOP algorithm.

B. Activation Functions

Activation functions are used in a neural network (NN) to introduce non-linearity that enables the model to learn more complex patterns [1]. Without activation functions, a NN can only seperate linear data, that limits the performance and capability of the model in most cases.

An activation function receives the net input signal and bias of the previous layer and determines the output of the neuron. The net input of the hidden layer is calculated with the equation below:

$$net_{y_{j,p}} = \sum_{i=1}^{I+1} v_{ji} z_{i,p} \tag{2}$$

where $net_{y_{j,p}}$ is the net input of the j-th neuron in the hidden layer for the p-th pattern.

The equation of the net input equation of the output layer is as follows:

$$net_{o_{k,p}} = \sum_{j=1}^{J+1} w_{kj} y_{j,p} \tag{3}$$

where $net_{o_{k,p}}$ is the net input of the k-th neuron in the output layer for the p-th pattern.

Several activation functions can be used when a FFNN model is constructed, such as the sigmoid activation function, the softmax activation function, the linear activation function and the rectified linear unit (RELU) activation function.

1) Sigmoid activation function: The sigmoid activation function is primarily used in the output layer for binary classification tasks. The equation used to calculate the sigmoid activation function of the output layer is as follows:

$$o_{k,p} = f_{o_k}(net_{o_{k,p}}) = \frac{1}{1 + e^{-net_{o_{k,p}}}} \tag{4}$$

In this case, the output layer contains only one neuron, so the only value that k can take is one.

The derivative of the sigmoid activation function of the output layer is given by the equation below:

$$f'_{o_k}(net_{o_{k,p}}) = (1 - o_{k,p}) o_{k,p} \tag{5}$$

where f'_{o_k} is the derivative of the activation function of the k-th output neuron.

2) Softmax activation function: The softmax activation function is commonly used in the output layer for multi-class classification tasks. To calculate the softmax activation function of the output layer, the equation below is used:

$$o_{k,p} = f_{o_k}(net_{o_{k,p}}) = \frac{e^{net_{o_{k,p}}}}{\Sigma_{l=1}^{K} e^{net_{o_{l,p}}}} \tag{6}$$

where K is the number of classes in the multi-class classifier. The softmax activation function returns K values that sums up to one and the pattern is then classified as the class corresponding to the largest output value.

3) Linear activation function: The linear activation function is mainly used for regression tasks in the output layer of the FFNN. The output of the FFNN model that uses a linear activation function in the output layer will result in a continuous value, that makes the FFNN model suitable to predict real-valued outcomes or for functions approximate tasks. To equation used to calculate the linear activation function of the output layer is as follows:

$$o_{k,p} = f_{o_k}(net_{o_{k,p}}) = net_{o_{k,p}} \tag{7}$$

In this case, the output layer contains only one neuron, so the only value that k can take is one.

4) RELU activation function: The RELU activation function is widely used in the hidden layers of a NN as it helps with faster convergence and reduces the likelihood of the problem where the gradient vanishes [7]. The equation used to calculate the RELU activation function of the hidden layer is as follows:

$$y_{j,p} = f_{y_j}(net_{y_{j,p}}) = max(0, net_{y_{j,p}}) \tag{8}$$

The derivative of the RELU activation function is expressed by use of the equation below:

$$f'_{y_j}(net_{y_{j,p}}) = \begin{cases} 1 & net_{y_{j,p}} > 0 \\ 0 & net_{y_{j,p}} \leq 0 \end{cases} \tag{9}$$

where f'_{y_j} is the derivative of the activation function of the j-th hidden neuron.

C. Objective Functions

An objective function measures the discrepancy between the actual target values of each pattern in a dataset and the output values produced by the FFNN model [3]. The FFNN model attempts to find the values of each weight from the input layer to the hidden layer, v_{ji}, and each weight from the hidden to the output layer, w_{kj}, that minimises the objective function of the model. A few examples of different types of objective functions is the binary cross-entropy error, the categorical cross-entropy error and the mean squared error (MSE).

1) Binary cross-entropy error: The binary cross-entropy objective function is primarily used for binary classification tasks and therefore it is a good choice to use this objective function when the sigmoid activation function is used as the activation function in the output layer of a FFNN model. The equation used to calculate the binary cross-entropy objective function for the FFNN output layer that employs a sigmoid activation function is as follows:

$$\mathcal{E} = -\frac{1}{P}\sum_{p=1}^{P}((t_{k,p}log(o_{k,p})) + (1 - t_{k,p})log(1 - o_{k,p})) \tag{10}$$

where $\mathcal{E}$ is the objective function of the FFNN, k can only be a value of 1 as there is only one neuron in the output layer, P is the number of patterns in the dataset and $t_{k,p}$ is the true target value of the p-th pattern.

The derivative of the binary cross-entropy objective function with respect to the net input signal for the FFNN output layer that employs a sigmoid activation function is calculated by the use of the equation below:

$$\delta_{o_{k,p}} = (o_{k,p} - t_{k,p})f'_{o_k}(net_{o_{k,p}}) \tag{11}$$

where $\delta_{o_{k,p}}$ is the error signal of the k-th neuron for the p-th pattern. $f'_{o_k}(net_{o_{k,p}})$ is calculated as shown in Equation 5.

2) Categorical cross-entropy error: The categorical cross-entropy objective function is typically used in multi-class classification tasks, which makes the objective function a good choice when the softmax activation function is deployed as the activation function for the output layer of a FFNN model. The equation used to calculate the categorical cross-entropy objective function for the FFNN output layer that employs a softmax activation function is as follows:

$$\mathcal{E} = -\frac{1}{P}\sum_{p=1}^{P}\sum_{k=1}^{K} t_{k,p}log(o_{k,p}) \tag{12}$$

The derivative of the categorical cross-entropy objective function with respect to the net input signal for the FFNN output layer, which uses a softmax activation function, can be calculated by use of the equation below:

$$\delta_{o_{k,p}} = o_{k,p} - t_{k,p} \tag{13}$$

3) MSE: The MSE objective function is commonly used for regression tasks, and it is a suitable choice when a linear activation function is employed in the output layer of a FFNN model. If the output layer of a FFNN model employs a linear activation function, then the equation used to calculate the MSE objective function is as follows:

$$\mathcal{E} = \frac{1}{P}\sum_{p=1}^{P}(t_{k,p} - o_{k,p})^2 \tag{14}$$

where k can only be a value of 1 as there is only one neuron in the output layer.

The derivative of the MSE objective function with respect to the net input signal for the FFNN output layer that employs a linear activation function is calculated by the use of the equation below:

$$\delta_{o_{k,p}} = \frac{2}{P}(t_{k,p} - o_{k,p}) \tag{15}$$

4) Error signal of the hidden layer: The hidden layer of the FFNN model employs a RELU activation function. To calculate the error signal for the hidden layer, the derivative of the objective function with respect to the net input signal for hidden layer is calculated. The equation follows as:

$$\delta_{y_{j,p}} = \sum_{k=1}^{K}\delta_{o_{k,p}}w_{kj}f'_{y_j}(net_{y_{j,p}}) \tag{16}$$

where $f'_{y_j}(net_{y_{j,p}})$ is calculated as shown in Equation 9.

D. Stochastic Gradient Descent

The concept of SGD was first introduced in 1951 when Sutton Monro and Herbert Robbins proposed the idea to update parameters iteratively by the use of randomly or noisy sampled data points [11]. In 1952, Jack Kiefer and Jacob Wolfowitz introduced the machine learning variant of SGD, that extended the ideas proposed by Monro and Robbins [8]. Kiefer and Wolfowitz developed an algorithm to estimate the minimum of an unknown function by iteratively adjusted parameters based on sampled data.

The SGD optimisation algorithm is a stochastic learning algorithm that can optimise a FFNN. Each iteration consists of two phases when the SGD optimisation algorithm is applied to a FFNN. The first phase is the feedforward pass phase, where the output values of the NN are calculated for each pattern used to construct the model. The second phase, known

as backward propagation, propagates the error signal from the output layer back toward the input layer. The weights are then adjusted as functions of the backpropagated error signal. The next epoch is executed with the updated weights. The SGD optimisation algorithm used in a FFNN with one hidden layer is represented by Algorithm 1

Algorithm 1 Stochastic Gradient Descent

Initialise weights, the learning rate η, the momentum α, and the number of epochs t = 0
while stopping condition(s) not true **do**
    Let $\mathcal{E}_T = 0$
    for each randomly selected training pattern p **do**
        Do the feedforward phase to calculate $y_{j,p}$ ($\forall\ j = 1, ..., J$) and $o_{k,p}$ ($\forall\ k = 1, ..., K$)
        Compute output error signals $\delta_{o_{k,p}}$ and hidden layer error signals $\delta_{y_{j,p}}$
        Adjust weights w_{kj} and v_{ji} (backpropagation of errors)
        $\mathcal{E}_T\ +=\ [\mathcal{E}_p = \Sigma_{k=1}^{K}(t_{k,p}log(o_{k,p}))]$
    end for
    $t = t + 1$
end while

The conditions used to stop the algorithm are:

- When the validation error, $\mathcal{E}_v$, reaches an acceptable level
- When the maximum number of epochs is exceeded
- When the model becomes prone to overfitting the data.
- When the average weight changes is very small

The equation used to adjust the weights that connects the input layer to the hidden layer is as follows:

$$v_{ji}(t) += \Delta v_{ji}(t) + \alpha \Delta v_{ji}(t-1) \tag{17}$$

where $alpha$ is the momentum of the FFNN model and the equation to calculate $\Delta v_{ji}(t)$ is as follows:

$$\Delta v_{ji}(t) = -\eta \delta_{y_{j,p}} z_{i,p} \tag{18}$$

where η is the learning rate of the FFNN model and $\delta_{y_{j,p}}$ is calculated as shown in Equation 16.

The equation used to adjust the weights that connects the hidden layer to the output layer is as follows:

$$w_{kj}(t) += \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1) \tag{19}$$

where the calculation of $\Delta w_{kj}(t)$ is given by the equation below:

$$\Delta w_{kj}(t) = -\eta \delta_{o_{k,p}} y_{j,p} \tag{20}$$

where the calculation for $\delta_{o_{k,p}}$ is shown in Equation 11 if the sigmoid activation function is employed in the output layer of the FFNN model. If the softmax activation function is employed, then $\delta_{o_{k,p}}$ is calculated as shown in Equation 13 and if the linear activation function is employed, $\delta_{o_{k,p}}$ is calculated as shown in Equation 16.

E. Scaled Conjugate Gradient

The SCG optimisation algorithm was introduced by Martin Fodslette Møller in 1993, with the goal to create a supervised learning algorithm that eliminates some of the disadvantages that most optimisation algorithms possess [10]. Specifically, SCG addresses issues such as the slow convergence rates and the dependency on the parameters that has to be specified by the user. Møller proposed the SCG algorithm as a batch learning algorithm, where the step sizes are automatically determined and the algorithm restarts if a good solution was not found. The SCG optimisation algorithm is summarised in Algorithm 2.

Algorithm 2 Scaled Conjugate Gradient

Initialise the weight vector $\mathbf{w}(1)$ and the scalars $\sigma > 0$, $\lambda_1 > 0$ and $\bar{\lambda} = 0$
Let $\mathbf{p}(1)$ = $\mathbf{r}(1)$ = -$\mathcal{E}'(\mathbf{w}(1))$, $t = 1$ and $success = true$
Label A: **if** $success = true$ **then**
    Calculate the second-order information
end if
Scale $\mathbf{s}(t)$ and $\delta(t)$
if $\delta(t) \leq 0$ **then**
    Make the Hessian matrix positive definite
end if
Calculate the step size
Calculate the comparison parameter
if $\Delta(t) \geq 0$ **then**
    A successful reduction in error can be made, so adjust the weights
    $\bar{\lambda}(t) = 0$
    $success = true$
    if $t\ mod\ n_w = 0$ **then**
        Restart the algorithm, with $\mathbf{p}(t+1) = \mathbf{r}(t+1)$ and go to label A
    else
        Create a new conjugate direction
    end if
    if $\Delta(t) \geq 0.75$ **then**
        Reduce the scale parameter with $\lambda(t) = \frac{1}{2}\lambda(t)$
    end if
else
    A reduction in error is not possible, so let $\bar{\lambda}(t) = \lambda(t)$ and $success = false$
end if
if $\Delta(t) < 0.25$ **then**
    Increase the scale parameter to $\lambda(t) = 4\lambda(t)$
end if
if $the\ steepest\ descent\ direction\ \mathbf{r}(t) \neq 0$ **then**
    Set $t = t + 1$ and go to label A
else
    Terminate and return $\mathbf{w}(t+1)$ as the desired minimum
end if

In the context of Algorithm 2, the number of weights within the FFNN model is denoted as n_w and $\mathbf{w}(t)$ is the weight

vector of the FFNN, that includes both of the hidden and output layer weights. Additionally, Algorithm 2 computes the step sizes automatically and restarts with a different search direction after n_w consecutive epochs if no reduction in error is achieved.

$\mathcal{E}'(\mathbf{w}(t))$ is the gradient of the objective function $\mathcal{E}$ with respect to each weight for each pattern at epoch t, and it is a vector that contains all of the error signals, $\delta_{o_{k,p}}$ and $\delta_{y_{j,p}}$. Here, $\delta_{o_{k,p}}$ is calculated as shown in Equation 11, Equation 13 or, Equation 15 if the output layer activation function is the sigmoid, softmax, or linear activation function, respectively and $\delta_{y_{j,p}}$ is calculated as shown in Equation 16.

The detailed steps of Algorithm 2 is outlined below:

- Calculation of second-order information:

$$\sigma(t) = \frac{\sigma}{\|\mathbf{p}(t)\|} \tag{21}$$

where $\sigma(t)$ is the second-order information at epoch t, $\mathbf{p}(t)$ is the search direction vector at epoch t and $\|\mathbf{p}(t)\|$ is the euclidean norm of vector $\mathbf{p}(t)$.

$$\mathbf{s}(t) = \frac{\mathcal{E}'(\mathbf{w}(t) + \sigma(t)\mathbf{p}(t)) - \mathcal{E}'(\mathbf{w}(t))}{\sigma(t)} \tag{22}$$

where $\mathbf{s}(t)$ is the approximation of the Hessian-vector product.

$$\delta(t) = \mathbf{p}(t)^T \mathbf{s}(t) \tag{23}$$

where $\mathbf{p}(t)^T$ is the transpose of vector $\mathbf{p}(t)$.

- Perform scaling:

$$\mathbf{s}(t) = \mathbf{s}(t) + (\lambda(t) - \bar{\lambda}(t))\mathbf{p}(t) \tag{24}$$

$$\delta(t) = \delta(t) + (\lambda(t) - \bar{\lambda}(t))\|\mathbf{p}(t)^2\| \tag{25}$$

- Make the Hessian matrix positive definite:

$$\mathbf{s}(t) = \mathbf{s}(t) + \left(\lambda(t) - 2\frac{\delta(t)}{\|\mathbf{p}(t)\|^2}\right)\mathbf{p}(t) \tag{26}$$

$$\bar{\lambda}(t) = 2\left(\lambda(t) - 2\frac{\delta(t)}{\|\mathbf{p}(t)\|^2}\right) \tag{27}$$

$$\delta(t) = -\delta(t) + \lambda(t)\|\mathbf{p}(t)\|^2 \tag{28}$$

$$\lambda(t) = \bar{\lambda}(t) \tag{29}$$

- Calculate the step size:

$$\mu(t) = \mathbf{p}(t)^T \mathbf{r}(t) \tag{30}$$

where $\mathbf{r}(t)$ is the negative gradient vector at epoch t.

$$\eta(t) = \frac{\mu(t)}{\delta(t)} \tag{31}$$

where $\eta(t)$ is the step size at epoch t.

- Calculate the comparison parameter:

$$\Delta(t) = \frac{2\delta(t)[\mathcal{E}(\mathbf{w}(t)) - \mathcal{E}(\mathbf{w}(t) + \eta(t)\mathbf{p}(t))]}{\mu(t)^2} \tag{32}$$

where $\Delta(t)$ is the comparison parameter.

- Adjust the weights:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta(t)\mathbf{p}(t) \tag{33}$$

$$\mathbf{r}(t+1) = -\mathcal{E}'(\mathbf{w}(t+1)) \tag{34}$$

- Create a new conjugate direction:

$$\beta(t) = \frac{\|\mathbf{r}(t+1)\|^2 - \mathbf{r}(t+1)^T\mathbf{r}(t)}{\mu(t)} \tag{35}$$

$$\mathbf{p}(t+1) = \mathbf{r}(t+1) + \beta(t)\mathbf{p}(t) \tag{36}$$

F. Leap Frog

The LFOP algorithm, introduced by Jan Snyman in 1982, is a dynamic optimisation algorithm designed to address unconstrained minimisation problems [13]. The LFOP algorithm employs a dynamic and adaptive approach that could efficiently search for the global minimum and not be constrained by the typical challenges posed by local minima. The LFOP algorithm is explained in Algorithm 3.

G. Performance Metrics

Performance metrics serve as essential tools for the evaluation of the effectiveness of classification and function approximation models. Performance metrics provide a quantitative measure of how reliably and accuratly a prediction model performs on a given task. Key metrics include accuracy and the MSE [4].

1) Accurcay: Accuracy is a common method used to evaluate the performance of classification models. The accuracy of a predictive classification model is determined by the proportion of correctly predicted labels against the total number of predictions. The calculation of the accuracy of a predictive model is as follows:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{37}$$

Accuracy is a popular choice of performance measure mainly beacause it is fairly easy to understand and compute and generally performs well on well balanced datasets.

2) Mean Squared Error: The MSE is commonly used as a performance metric in function approximation tasks. The MSE quantifies the average squared difference between predicted and actual values, that provides insights into model accuracy [2]. Additionally, the MSE effectively identifies larger errors, which is crucial for FFNN models due to their sensitivity to outliers. The calculation of the MSE for the FFNN model is performed by use of Equation 14.

Algorithm 3 Leap Frog

Create a random initial solution $\mathbf{w}(0)$, and let $t = -1$
Let $\Delta t = 0.5, \delta = 1, m = 3, \delta_1 = 0.001, \epsilon = 10^{-5}, i = 0, j = 2, s = 0, p = 1$
Compute the initial acceleration $\mathbf{a}(0) = -\mathcal{E}'(\mathbf{w}(0))$ and velocity $\mathbf{v}(0) = \frac{1}{2}\mathbf{a}(0)\Delta t$
repeat
    $t = t + 1$
    Compute $\|\Delta\mathbf{w}(t)\| = \|\mathbf{v}(t)\|\Delta t$
    if $\|\Delta\mathbf{w}(t)\| < \delta$ **then**
        $p = p + \delta_1, \Delta t = p\Delta t$
    else
        $\mathbf{v}(t) = \delta\mathbf{v}(t)/(\Delta t\|\mathbf{v}(t)\|)$
    end if
    if $s \geq m$ **then**
        $\Delta t = \Delta t/2, s = 0$
        $\mathbf{w}(t) = (\mathbf{w}(t) + \mathbf{w}(t-1))/2$
        $\mathbf{v}(t) = (\mathbf{v}(t) + \mathbf{v}(t-1))/4$
    end if
    $\mathbf{w}(t+1) = \mathbf{w}(t) + \mathbf{v}(t)\Delta t$
    repeat
        $\mathbf{a}(t+1) = -\mathcal{E}'(\mathbf{w}(t+1))$
        $\mathbf{v}(t+1) = \mathbf{v}(t) + \mathbf{a}(t+1)\Delta t$
        if $\mathbf{a}^T(t+1)\mathbf{a}(t) > 0$ **then**
            $s = 0$
        else
            $s = s + 1, p = 1$
        end if
        if $\|\mathbf{a}(t+1)\| > \epsilon$ **then**
            if $\|\mathbf{v}(t+1)\| > \|\mathbf{v}(t)\|$ **then**
                $i = 0$
            else
                $\mathbf{w}(t+2) = (\mathbf{w}(t+1) + \mathbf{w}(t))/2$
                $i = i + 1$
                Perform a restart: **if** $i \leq j$ **then**
                $\mathbf{v}(t+1) = (\mathbf{v}(t+1) + \mathbf{v}(t))/4$
                $t = t + 1$
                else
                $\mathbf{v}(t+1) = 0, j = 1, t = t + 1$
                end if
            end if
        end if
    until $\|\mathbf{v}(t+1)\| > \|\mathbf{v}(t)\|$
until $\|\mathbf{a}(t+1)\| \leq \epsilon$
Return $\mathbf{w}(t)$ as the solution

H. Weight Decay

John Hertz and Anders Krogh published a paper in 1991, that explains why weight decay improves the generalisation in a FFNN [5]. Hertz and Krogh proved that the addition of weight decay suppresses any irrelevant components of the weight vector by selecting the smallest vector that solves the learning problem. Weight decay also reduces the effects of static noise on the targets, which improves generalisation quite a lot.

Weight decay prevents large weights and is implemented by the addition of a penalty term to the objective function that penalises large weights. The mathematical equation of the addition of weight decay is represented by the equation below:

$$\mathcal{E}(\mathbf{w}) = \mathcal{E}_T(\mathbf{w}) + \frac{1}{2}\lambda\sum_{i=1}^{n_w} w_i^2 \tag{38}$$

where $\mathcal{E}_T(\mathbf{w})$ is calculated as shown in Equation 10, Equation 12 or Equation 14, if the output layer activation function is the sigmoid, softmax or linear activation function, respectively with respect to the weights $\mathbf{w}$ and λ is a parameter that determines the strength of the penalty on large weights.

Weight decay also has to be taken into consideration in the backpropagation step when the gradient of the FFNN objective function is calculated. The equation used to incorporate the weight decay in the backpropagation of the FFNN model is as follows:

$$\mathcal{E}'(\mathbf{w}) = -\mathcal{E}_T'(\mathbf{w}) - \lambda\mathbf{w} \tag{39}$$

III. Implementation

This section provides the approach taken to implement a FFNN model with one hidden layer, as well as the integration of the SGD, SCG, and LFOP optimisation algorithms, for both classification and function approximation tasks.

A. Feed Forward Neural Network Architecture

Each FFNN model is implemented as outlined in Section II-A. The architecture consists of an input layer, a single hidden layer, and an output layer.

The RELU activation function is employed as the activation function in the hidden layer. Additionally, the output layer employs different activation functions based on the task:

- The sigmoid activation function for binary classification tasks.
- The softmax activation function for multi-class classification tasks.
- The linear activation function for function approximation tasks.

These activation functions are implemented as described in Section II-B.

The objective function used by the FFNN model depends on the activation function employed in the output layer:

- The binary cross-entropy when the sigmoid activation function is employed.
- The categorical cross-entropy when the softmax activation function is employed.
- The MSE when the linear activation function is employed.

These objective functions are implemented as described in Section II-C.

Each FFNN model implements early stopping. The model stops if the maximum number of epochs is exceeded, when the average weight change is less than 10^{-6}, and when overfitting is detected. The model is deemed to overfit when the validation

error increases, while the training error decreases. The equation used to ensure that the model does not stop immediatly when the validation error increases, as it might decrease on the next epoch, is as follows:

$$\mathcal{E}_v > \bar{\mathcal{E}}_v + \sigma_{\mathcal{E}_v} \tag{40}$$

where $\mathcal{E}_v$ is the moving average of the validation error calculated on each new epoch, and $\sigma_{\mathcal{E}_v}$ is the standard deviation of the validation errors. If any of these early stopping techniques is triggered, the weight vectors are restored to where the validation error was best.

Weight decay is also implemented in each FFNN, as described in Section II-H, to remove irrelevant and redundant weights.

The weights of the FFNN model are sampled from a uniform distribution function to initialise all weights from the input to the hidden layer and all weights from the hidden to the output layer. The equation used to sample the initial weights from a uniform distribution is as follows:

$$w_{kj}, v_{ji} \sim Uniform\left(\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}\right) \tag{41}$$

where $fanin$ is the number of connections that leads into the neuron. Therefore, $fanin$ would be $I+1$ when v_{ji} is initialised and $J+1$ when w_{kj} is initialised.

B. Stochastic Gradient Descent Optimisation Algorithm

The SGD optimisation algorithm is implemented as described in Algorithm 1. Additionally, Equations 17-20 is utilised to further refine the implementation of the SGD algorithm.

C. Scaled Conjugate Gradient Optimisation Algorithm

The SCG optimisation algorithm is implemented as described in Algorithm 2. Furthermore, Equations 21-36 is incorporated to enhance the implementation of the SCG optimisation algorithm.

D. Leap Frog Optimisation Algorithm

The LFOP algorithm is implemented as described in Algorithm 3. Additionally, a process of mini-batch learning is used to improve the efficiency and stability of the model. This approach divides the training dataset into smaller subsets, known as mini-batches, and allows the model to update its weights more frequently, which often leads to faster convergence and improved performance on larger datasets. The implementation of the LFOP algorithm with the use of mini-batch learning is provided in Algorithm 4.

Algorithm 4 Leap Frog

Create a random initial solution $\mathbf{w}(0)$, let $t = -1$, and initialise the number of batches $n_b \geq 1$
Let $\Delta t = 0.5, \delta = 1, m = 3, \delta_1 = 0.001, \epsilon = 10^{-5}, i = 0, j = 2, s = 0, p = 1$
Compute the initial acceleration $\mathbf{a}(0) = -\mathcal{E}'(\mathbf{w}(0))$ and velocity $\mathbf{v}(0) = \frac{1}{2}\mathbf{a}(0)\Delta t$
repeat
    $t = t + 1$
    Compute $\|\Delta\mathbf{w}(t)\| = \|\mathbf{v}(t)\|\Delta t$
    if $\|\Delta\mathbf{w}(t)\| < \delta$ **then**
        $p = p + \delta_1, \Delta t = p\Delta t$
    else
        $\mathbf{v}(t) = \delta\mathbf{v}(t)/(\Delta t\|\mathbf{v}(t)\|)$
    end if
    if $s \geq m$ **then**
        $\Delta t = \Delta t/2, s = 0$
        $\mathbf{w}(t) = (\mathbf{w}(t) + \mathbf{w}(t-1))/2$
        $\mathbf{v}(t) = (\mathbf{v}(t) + \mathbf{v}(t-1))/4$
    end if
    $\mathbf{w}(t+1) = \mathbf{w}(t) + \mathbf{v}(t)\Delta t$
    repeat
        Divide the dataset into n_b batches
        Initialise an empty list batch_gradients
        for each batch **do**
            Compute gradient for the batch: $\mathcal{E}'_{\text{batch}}(\mathbf{w}(t+1))$
            Append the batch gradient to batch_gradients
        end for
        Compute the mean gradient: $\overline{\mathcal{E}'}(\mathbf{w}(t+1)) = \frac{1}{n_b}\sum_{\text{batch}}\mathcal{E}'_{\text{batch}}(\mathbf{w}(t+1))$
        $\mathbf{a}(t+1) = -\overline{\mathcal{E}'}(\mathbf{w}(t+1))$
        $\mathbf{v}(t+1) = \mathbf{v}(t) + \mathbf{a}(t+1)\Delta t$
        if $\mathbf{a}^T(t+1)\mathbf{a}(t) > 0$ **then**
            $s = 0$
        else
            $s = s + 1, p = 1$
        end if
        if $\|\mathbf{a}(t+1)\| > \epsilon$ **then**
            if $\|\mathbf{v}(t+1)\| > \|\mathbf{v}(t)\|$ **then**
                $i = 0$
            else
                $\mathbf{w}(t+2) = (\mathbf{w}(t+1) + \mathbf{w}(t))/2$
                $i = i + 1$
                Perform a restart: **if** $i \leq j$ **then**
                $\mathbf{v}(t+1) = (\mathbf{v}(t+1) + \mathbf{v}(t))/4$
                $t = t + 1$
                else
                $\mathbf{v}(t+1) = 0, j = 1, t = t + 1$
                end if
            end if
        end if
    until $\|\mathbf{v}(t+1)\| > \|\mathbf{v}(t)\|$
until $\|\mathbf{a}(t+1)\| \leq \epsilon$
Return $\mathbf{w}(t)$ as the solution

IV. Empirical Procedure

This section of the report outlines the systematic approach used to evaluate the performance of the FFNN model that employs the SGD, SCG, and LFOP optimisation algorithms across multiple classification and function approximation tasks. Additionally, this section provides the approach used to process each dataset to ensure optimal performance of the FFNN model. The section provides the performance metrics used to compare the performance of the optimisation algorithms, the control parameters used to construct each FFNN model for each of the classification and function approximation tasks, the experimental setup and the statistical significance tests used to ensure the reliability and statistical soundness of the results.

A. Performance Metrics

Accuracy is utilised to evaluate and compare the performance of each FFNN model across all three classification tasks, which should be maximised. Additionally, all three function approximation tasks are evaluated and compared by the use of the MSE, which should minimised. The time taken for each model construction, measured in seconds, is also considered for comparison.

B. Data Preprocessing

To ensure optimal and reliable results on each dataset from each FFNN model, the data has to be pre-processed. The preprocessing procedures differ between each dataset, and therefore must each dataset be explored separately and processed into an optimal form to use in the FFNN model.

1) Breast Cancer Dataset: The breast cancer dataset contains numerical features computed from a digitised image of a fine needle aspirate (FNA) of a breast mass, that describe characteristics of the cell nuclei present in the image. The dataset includes a binary target feature labeled 'diagnosis,' which indicates whether the tumor is benign or malignant, denoted by 'B' or 'M', respectively. In total, the dataset consists of 33 features and 569 observations.

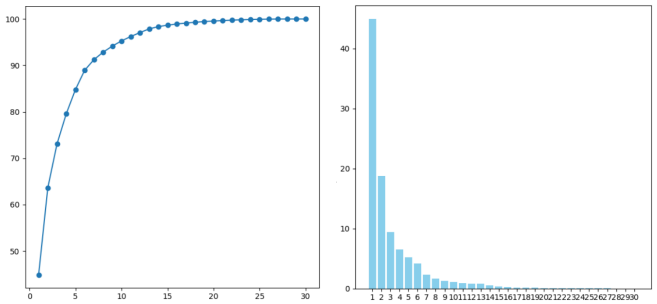
The dataset contains a unique identification feature 'id' and an unknown feature 'Unnamed: 32' that is removed from the dataset, as it does not provide meaningful infomation. There are no missing values or outliers in the dataset. However, the classes are unbalanced, therefore the data is resampled by use of the synthetic minority oversampling technique (SMOTE) to ensure balanced classes.

All of the features that remain are numerical and are scaled to have a mean of zero and a standard deviation of one to ensure optimal performance of the FFNN model. The equation used to scale the data is as follows:

$$Z_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j} \tag{42}$$

where $x_{i,j}$ is the data value of the j-th feature for the i-th observation, μ_j and σ_j is the mean and standard deviation of the j-th feature across all observations, respectively, and $Z_{i,j}$ is the standardised data value of the data value at the i-th observation and the j-th feature.

The principal component analysis (PCA) technique is employed to reduce the dimensionality of the dataset. The first 10 principle components captures 95.28% of the cumulative explained variance as shown in Figure 1a and Figure 1b shows that from principal component six and onwards captures less than 5% varaince each.



(a) Cumulative Explained Variance by PCA Components (b) Explained Variance by Each Principal Component

Fig. 1: Principal Component Analysis (PCA) Variance Explanation of the breast cancer dataset

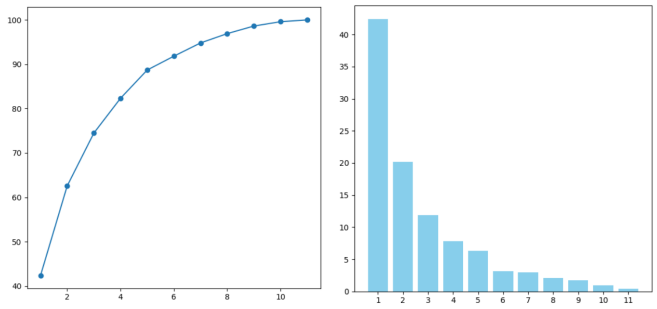
The first 10 principal components are chosen as the features, which decreases the number of features from 30 to 10.

The target feature 'diagnosis' is encoded so that each instance of 'M' is represented as a zero and each instance of 'B' is represented as one. Additionally, the sigmoid activation function is employed in the output layer.

2) Body Performance Dataset: The body performance dataset contains various fitness and health metrics collected from individuals. The target feature, 'class', categorises individuals into one of four classes: A, B, C or D. The individuals in class A are deemed to be the fittest. In total, the dataset consists of 12 features and 13393 observations.

There are no unique features or missing values in the dataset and the classes are all balanced. There is a categorical feature, 'gender', in the body performance dataset, that contains an 'M' to indicate male and a 'F' to indicate female. This categorical feature is encoded so that each instance of a 'M' is represented by zero and each insatnce of a 'F' is represented by one. The remainder of the features are continuous features, so each continuous feature should be scaled by use of Equation 42.

The PCA technique is utilised to reduce the dimensions of the dataset. The first eight principle components captures 96.91% of the cumulative explained variance as shown in Figure 2a and Figure 2b shows that from principal component six and onwards captures less than 5% varaince each.



(a) Cumulative Explained Variance by PCA Components (b) Explained Variance by Each Principal Component

Fig. 2: Principal Component Analysis (PCA) Variance Explanation of the body performance dataset

The first eight principal components are chosen as the features, which decreases the number of features from 11 to eight.

The target feature is one-hot encoded, which converts each class label into a binary vector representation. Additionally, the softmax activation function is employed in the output layer.

3) Fashion MNIST Dataset: The fashion MNIST dataset is a collection of grayscale images, where each image has 28 pixels in height and 28 pixels in width. The pixel values are integers between zero and 255, with each pixel associated with one pixel value. A higher value indicates a lighter pixel. In total, the dataset consists of 784 features and 70000 observations, where the 28×28 pixels are flattened to create a vetor of pixels, that results in 784 features.

The fashion MNIST dataset includes 10 different clothes categories. These categories are: 'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', and 'Ankle boot'.

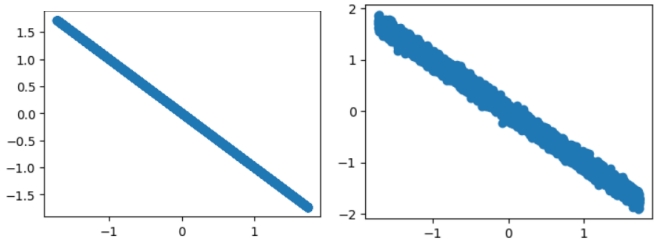
There are no missing values or imbalanced classes. However, the data is scaled by dividing each pixel value by 255 to obtain values between zero and one. The target feature is one-hot encoded, which converts each class label into a binary vector representation. Additionally, the softmax activation function is employed in the output layer.

4) Linear function: The first function to be approximated is a linear function represented by the equation below:

$$y = -1.2x + 4.2 + err \tag{43}$$

where y is the function to approximate, x is a random value generated from a uniform distribution between -10 and 10, that is $x \sim Uniform(-10, 10)$ and e is an error generated from a normal distribution with a mean of zero and a standard deviation of 0.5, that is $err \sim Normal(0, 0.5)$. Additionally, the linear activation function is employed in the output layer. The introduction of err adds noise to the generated data to prevent the FFNN models from overfitting the training data. Both x and y are then scaled by use of Equation 42.

The scaled linear function can be seen in Figure 3a, while the scaled linear function with added error to introduce noise is displayed in Figure 3b.



(a) Scaled linear function (b) Scaled linear function with error

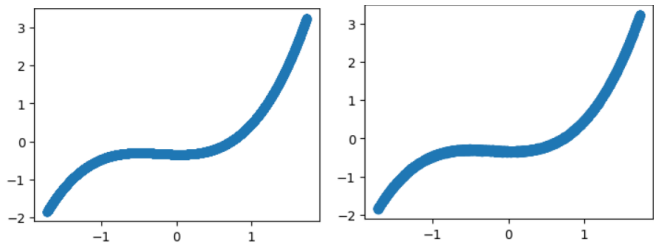
Fig. 3: Linear function to approximate

5) Cubic function: The second function to be approximated is a cubic function represented by the equation below:

$$y = 2x^3 + 4x^2 - x + 1 + err \tag{44}$$

where $x \sim Uniform(-5, 5)$ and $err \sim Normal(0, 1)$. Both x and y are then scaled by use of Equation 42. Additionally, the linear activation function is employed in the output layer.

The scaled cubic function can be seen in Figure 4a, while the scaled cubic function with added error to introduce noise is displayed in Figure 4b.



(a) Scaled cubic function (b) Scaled cubic function with error

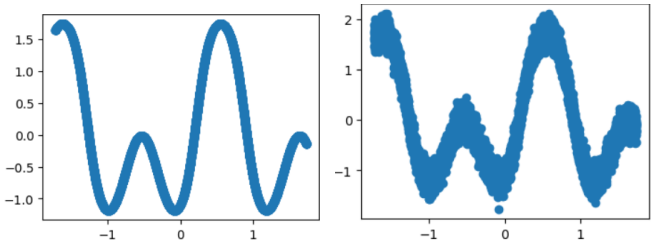
Fig. 4: Cubic function to approximate

6) Logistic function: The final function to be approximated is a logistic function represented by the equation below:

$$y = \frac{1}{1 + e^{-sin(x)+cos(2x)}} + err \tag{45}$$

where $x \sim Uniform(-5, 5)$ and $err \sim Normal(0, 0.03)$. Both x and y are then scaled by use of Equation 42. Additionally, the linear activation function is employed in the output layer.

The scaled logistic function can be seen in Figure 5a, while the scaled logistic function with added error to introduce noise is displayed in Figure 5b.



(a) Scaled Logistic function (b) Scaled Logistic function with error

Fig. 5: Logistic function to approximate

C. Experimental Setup

1) Comparison of optimisation algorithms: A 10-fold cross validation was used to evaluate the performance of each optimisation algorithm for both classification and function approximation tasks. For the classification tasks, the cross validation splits the data up into 10 datasets, where eight sets were used to construct the model, one set was used as the validation set and one set was used as the test set. This process was repeated 10 times to ensure each subset is used as the validation set once. The utilisation of the 10-fold cross validation technique ensured that the evaluation was fair and consistent across the entire dataset and the bias that might arise from relying on a single training-test split was reduced.

A 10-fold cross validation was used to evaluate the performance of each optimisation algorithm for each function approximation task. Each fold generated 10000 instances used to construct each model, 2000 instances used as the validation set, and 2000 instances used as the test set. A random seed is initialised to ensure reproducibility of each generated dataset.

2) Finding the optimal control parameters: To find the optimal control parameters of each optimisation algorithm for each task, a 5-fold cross validation was used, as described above, for both the classification and function approximation tasks. A Bayesian optimisation technique efficiently searched the hyperparameter space to identify the parameters that minimised the average test accuracy error for classification tasks and the average MSE for function approximation tasks across the 5-fold cross validation.

D. Control Parameters

Bayesian optimisation is used to find the optimal control parameter for each classification and function approximation tasks, as described in Section IV-C. The optimal control parameters used for each SGD optimisation algorithm in both classification and function approximation tasks are represented in Table I.

The optimal control parameters used for each SCG optimisation algorithm in both classification and function approximation tasks are represented in Table II.

The optimal control parameters used for each LFOP optimisation algorithm in both classification and function approximation tasks are represented in Table III.

E. Statistical Significance and Analysis

The statistical significance of the results from the comparison of optimisation algorithms is determined by the evaluation of the accuracy for the classification tasks and the MSE for the function approximation tasks, by the utilisation of a 10-fold cross validation. The Kruskal-Wallis test is employed to check for stochastic dominance among the samples, with a significance level of 0.05.

V. Research Results

This section presents an overview of the experimental results obtained by the FFNN model that deploys the SGD, SCG, and LFOP optimisation algorithms.

A. Breast Cancer Dataset

By use of Bayesian optimisation, the optimal values of the control parameters in the SGD, SGD, and LFOP optimisation algorithm are determined as seen in Section IV-D.

A 10-fold cross validation technique is then applied to each of these FFNN models to compare each optimisation algorithm with one another. Table IV presents the average and standard deviation of the performance metrics of the 10-fold cross validation for each optimisation algorithm.

TABLE IV: Performance Metrics of each Optimisation Algorithm Obtained from Classifying the Breast Cancer Dataset

Algorithm	**Performance metrics**			
	mean accuracy	***std accuracy***	***mean time***	***std time***
SGD	96.91	0.0485	2.0621	2.150
SCG	97.47	0.0451	0.1211	0.075
LFOP	97.33	0.0408	1.6545	0.829

The accuracy scores obtained from each fold by each FFNN model are presented in Figure 6.

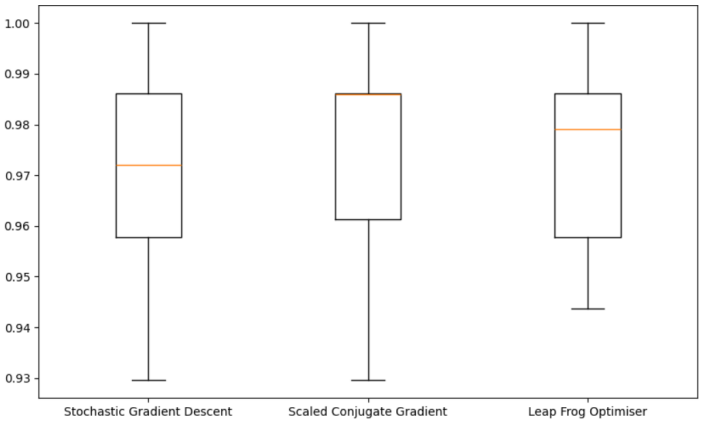


Fig. 6: Comparison of accuracy for three optimisation algorithms when the Breast Cancer dataset is classified by use of 10-fold cross-validation.

The Kruskal-Wallis statistical test returned a p-value of 0.94, which shows that there is no significant difference between the scores of each optimisation algorithm. However, the

TABLE I: Stochastic Gradient Descent Control Parameters

Dataset	Control Parameters					
	eta	*alpha*	*lambda*	*epochs*	*hidden units*	*Bias*
Breast Cancer	0.09298	0.9662	0.0001	56	9	-0.14345
Body Performance	0.00142	0.46241	4.18×10^{-6}	100	14	1
Fashion MNIST	0.00394	0.05615	2.31×10^{-5}	15	104	-0.04005
Linear Function	0.01	0	0.0001	100	237	-1
Cubic Function	0.01	0.53969	0.0001	100	424	-1
Logistic Function	0.00021	0.93863	0.0001	100	342	-1

TABLE II: Scaled Conjugate Gradient Control Parameters

Dataset	Control Parameters			
	lambda	*epochs*	*hidden units*	*Bias*
Breast Cancer	0.0001	10000	8	0.70733
Body Performance	3.91×10^{-6}	4027	32	-1
Fashion MNIST	9.2×10^{-5}	420	86	0.29634
Linear Function	0.0001	1000	69	0.25951
Cubic Function	1×10^{-6}	344	236	1
Logistic Function	1×10^{-6}	473	454	0.95896

TABLE III: Leap Frog Optimisation Control Parameters

Dataset	Control Parameters				
	lambda	*epochs*	*hidden units*	*Batches*	*Bias*
Breast Cancer	0.00801	4561	5	32	-1
Body Performance	8.22×10^{-5}	2000	21	1	-1
Fashion MNIST	0.00116	227	126	42	0.30628
Linear Function	0.0001	100	30	128	-0.02609
Cubic Function	0.0001	100	363	128	1
Logistic Function	0.0001	370	390	128	0.94595

Kruskal-Wallis statistical test returned a p-value of 6.23×10^{-5} between the construction times of the FFNN model, which shows a significant difference between the times of each optimisation algorithm. Figure 7 presents the average time each model takes for construction across the 10-folds.

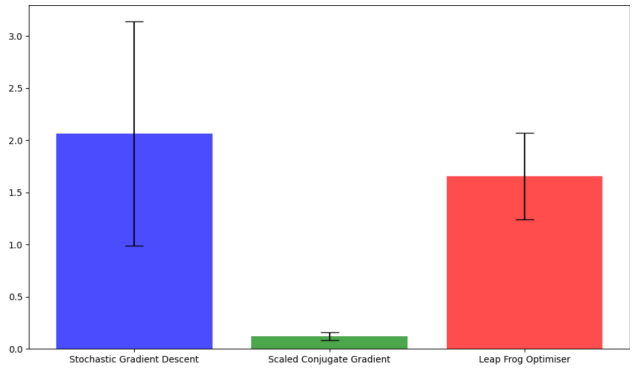
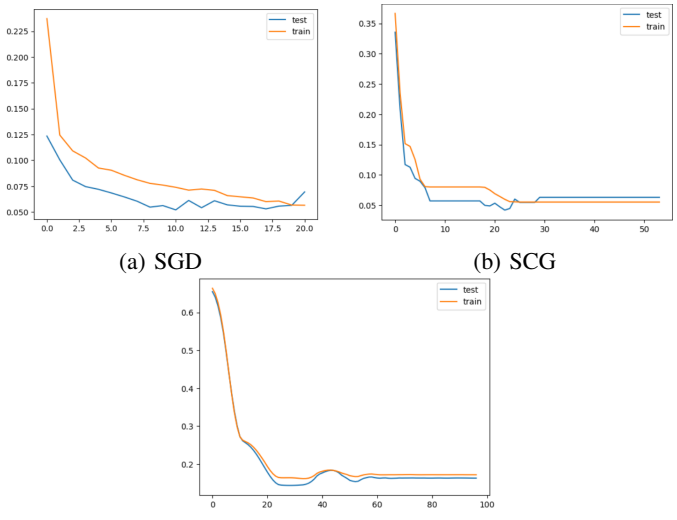


Fig. 7: Comparison of construction time for three optimisation algorithms when the Breast Cancer dataset is classified by use of 10-fold cross-validation.

The models are then constructed on the same data split. Figure 8 represents the training and validation errors over epochs for each optimization algorithm on this data split.



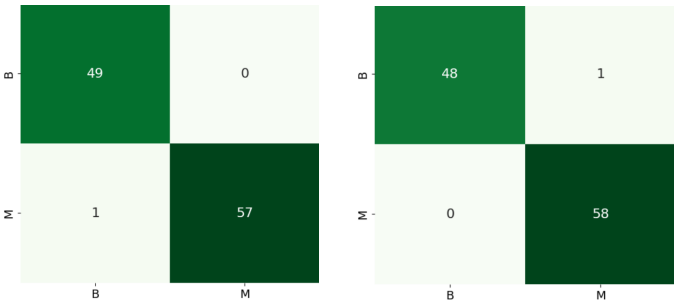
(a) SGD (b) SCG

(c) LFOP

Fig. 8: Training and validation errors over epochs for the Breast Cancer classification task of each optimisation algorithm

Both the SCG and LFOP algorithms does not overfit the training data, as seen in Figure 8b and 8c. However, the SGD optimisation algorithm does overfit to the training data, where early stopping is then triggered due to a validation error that increases, which can be seen in Figure 8a.

From this data split, the confusion matrices are produced to show the performance of each optimisation algorithm on the breast cancer dataset. These confusion matrices are presented in Figure 9.



(a) SGD (b) SCG

(c) LFOP

Fig. 9: Confusion matrix of the breast cancer dataset for each optimization algorithm, with true labels on the y-axis and predicted labels on the x-axis.

After the examination of the results given above, it is deduced that the best optimisation algorithm to use on the breast cancer dataset is the SCG optimisation algorithm. Although the LFOP algorithm has a smaller deviation between accuracies, the SCG optimisation algorithm has a greater mean accuracy and constructs the FFNN model the quickest. The confusion matrix in Figure 9b also shows that the incorrect prediction of the test set is the prediction that the instance is malignant when the true label is benign. When the context is considered, an incorrect prediction of malignant is a much better outcome than an incorrect prediction of benign. Therefore, the best optimisation algorithm to use when the breast cancer dataset is classified with a FFNN model, is the SCG optimisation algorithm.

B. Body Performance Dataset

By use of Bayesian optimisation, the optimal values of the control parameters in the SGD, SGD, and LFOP optimisation algorithm are determined as seen in Section IV-D.

A 10-fold cross validation technique is then applied to each of these FFNN models to compare each optimisation algorithm with one another. Table V presents the average and standard deviation of the performance metrics of the 10-fold cross validation for each optimisation algorithm.

TABLE V: Performance Metrics of each Optimisation Algorithm Obtained from Classifying the Body Performance Dataset

Algorithm	**Performance metrics**			
	mean accuracy	***std accuracy***	***mean time***	***std time***
SGD	69.89	0.0145	263.192	6.709
SCG	70.86	0.0096	231.208	120.219
LFOP	71.21	0.0143	35.172	13.186

The accuracy scores obtained from each fold by each FFNN model are presented in Figure 10.

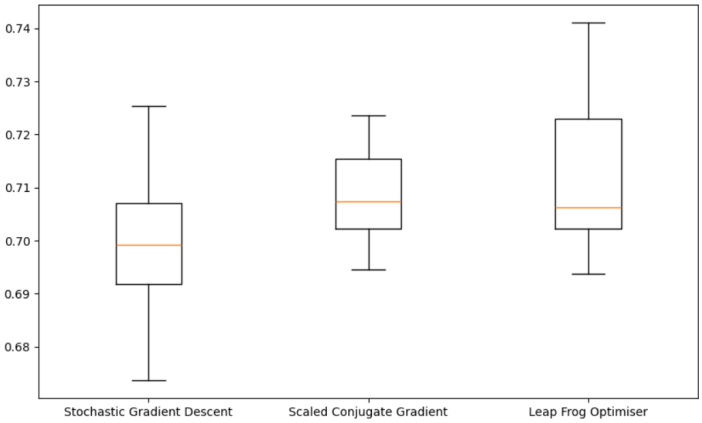


Fig. 10: Comparison of accuracy for three optimisation algorithms when the Body Performance dataset is classified by use of 10-fold cross-validation.

The Kruskal-Wallis statistical test returned a p-value of 0.184, which shows that there is no significant difference between the scores of each optimisation algorithm. However, the Kruskal-Wallis statistical test returned a p-value of 0.00029 between the construction times of the FFNN model, which shows a significant difference between the times of each optimisation algorithm. Figure 11 presents the average time each model takes for construction across the 10-folds.

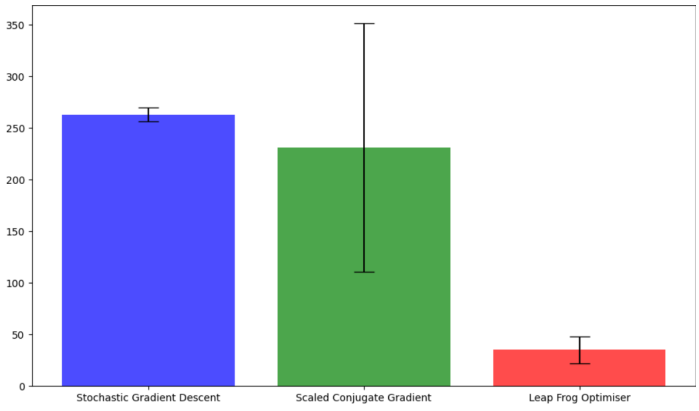
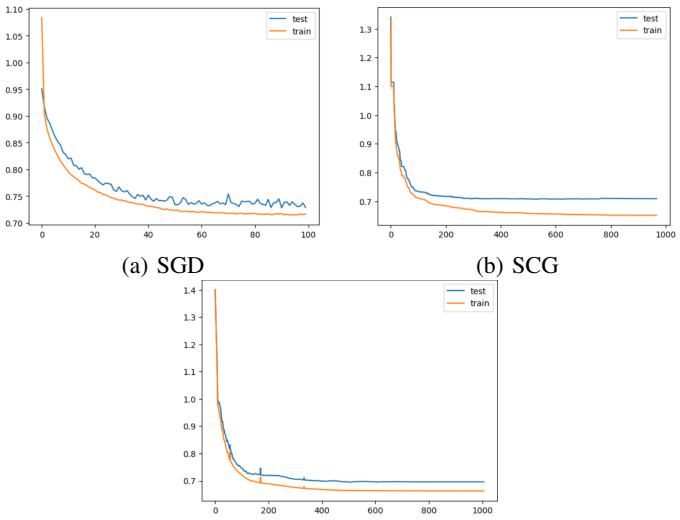


Fig. 11: Comparison of construction time for three optimisation algorithms when the Body Performance dataset is classified by use of 10-fold cross-validation.

The models are then constructed on the same data split. Figure 12 represents the training and validation errors over epochs for each optimization algorithm on this data split.



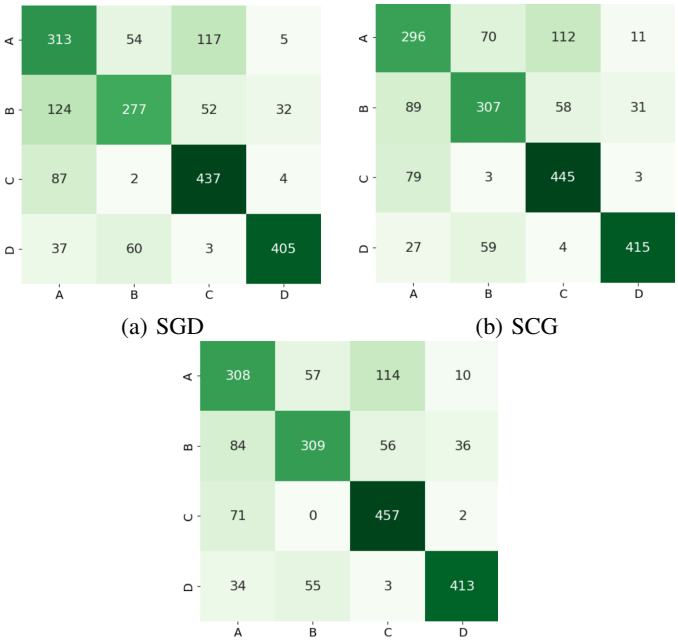
(a) SGD (b) SCG

(c) LFOP

Fig. 12: Training and validation errors over epochs for the Body Performance classification task of each optimisation algorithm

Both the SGD and LFOP algorithms does not overfit the training data, as seen in Figure 12a and 12c. However, the SCG optimisation algorithm does overfit to the training data, where early stopping is then triggered due to a validation error that increases, which can be seen in Figure 12b.

From this data split, the confusion matrices are produced to show the performance of each optimisation algorithm on the body performance dataset. These confusion matrices are presented in Figure 13.



(a) SGD (b) SCG

(c) LFOP

Fig. 13: Confusion matrix of the body performance dataset for each optimization algorithm, with true labels on the y-axis and predicted labels on the x-axis.

After the examination of the results given above, it is deduced that the best optimisation algorithm to use on the body performance dataset is the LFOP optimisation algorithm. The LFOP algorithm converges the quickest and obtained the largest mean accuracy score. The FFNN model also does not overfit on the training data when the LFOP algorithm is deployed, as shown by Figure 12c.

C. *Fashion MNIST Dataset*

By use of Bayesian optimisation, the optimal values of the control parameters in the SGD, SGD, and LFOP optimisation algorithm are determined as seen in Section IV-D.

A 10-fold cross validation technique is then applied to each of these FFNN models to compare each optimisation algorithm with one another. Table VI presents the average and standard deviation of the performance metrics of the 10-fold cross validation for each optimisation algorithm.

TABLE VI: Performance Metrics of each Optimisation Algorithm Obtained from Classifying the Fashion MNIST Dataset

Algorithm	**Performance metrics**			
	mean accuracy	***std accuracy***	***mean time***	***std time***
SGD	88.12	0.005	4129.87	1306.38
SCG	87.07	0.0064	6424.27	2932.92
LFOP	87.02	0.0033	1110.35	18.42

The accuracy scores obtained from each fold by each FFNN model are presented in Figure 14.

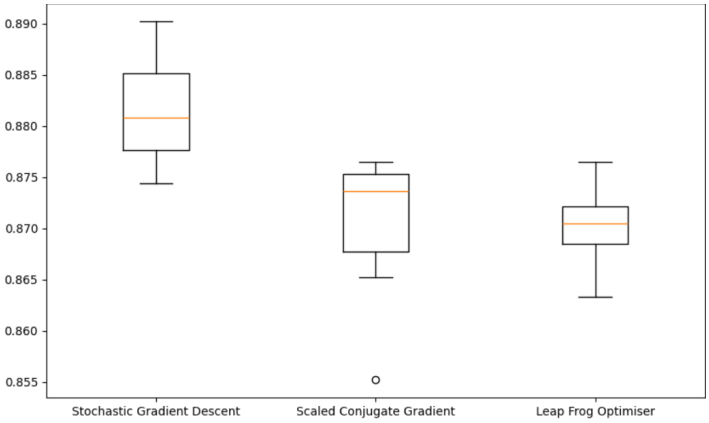


Fig. 14: Comparison of accuracy for three optimisation algorithms when the Fashion MNIST dataset is classified by use of 10-fold cross-validation.

The Kruskal-Wallis statistical test returned a p-value of 0.00041, which shows a significant difference between the scores of each optimisation algorithm. Figure 15 presents the average time each model takes for construction across the 10-folds.

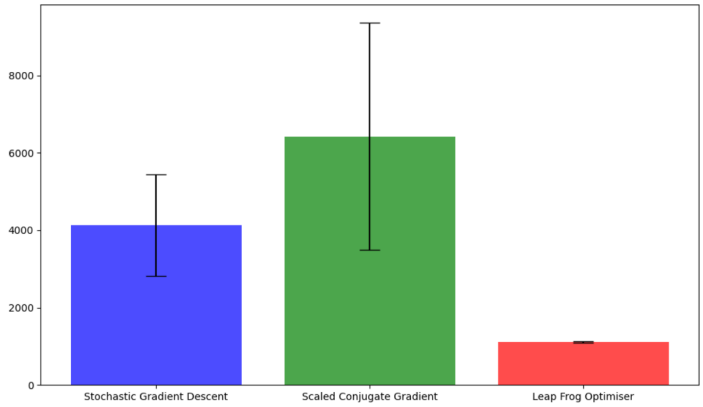
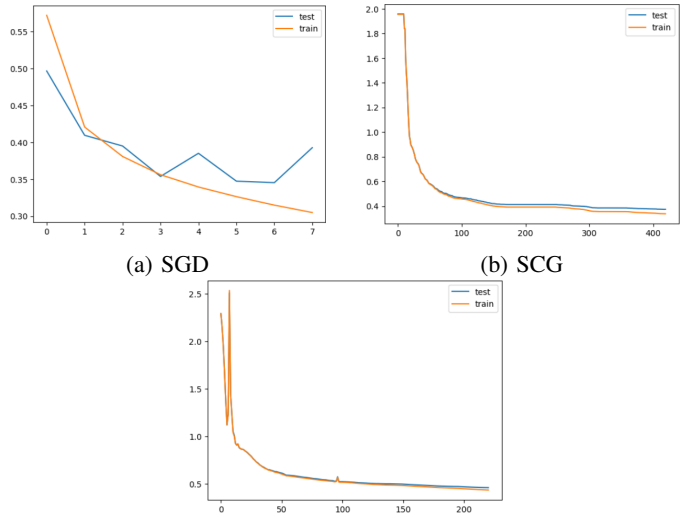


Fig. 15: Comparison of construction time for three optimisation algorithms when the Fashion MNIST dataset is classified by use of 10-fold cross-validation.

The models are then constructed on the same data split. Figure 16 represents the training and validation errors over epochs for each optimization algorithm on this data split.



(a) SGD (b) SCG

(c) LFOP

Fig. 16: Training and validation errors over epochs for the Fashion MNIST classification task of each optimisation algorithm

Both the SCG and LFOP algorithms does not overfit the training data, as seen in Figure 16b and 16c. However, the SGD optimisation algorithm does overfit to the training data, where early stopping is then triggered due to a validation error that increases, which can be seen in Figure 16a.

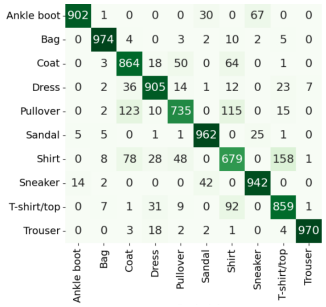
From this data split, the confusion matrices are produced to show the performance of each optimisation algorithm on the fashion MNIST dataset. These confusion matrices are presented in Figure 17.

After the examination of the results given above, it is deduced that the best optimisation algorithm to use on the fashion MNIST dataset is the SGD optimisation algorithm. The SGD algorithm the largest mean accuracy score and has the best distribution of accuracy scores. The SGD optimisation algorithm is generally good at dealing with complex data patters, and although the algorithm takes longer to converge than the LFOP algorithm, the accuracy scores obtained by the SGD model makes it the best optimisation algorithm for the fashion MNIST dataset.

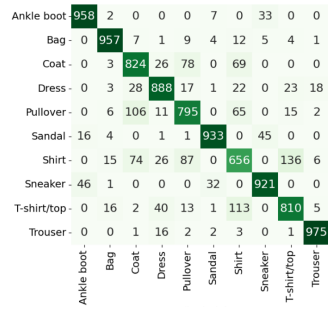
D. Linear Function Approximation

By use of Bayesian optimisation, the optimal values of the control parameters in the SGD, SGD, and LFOP optimisation algorithm are determined as seen in Section IV-D.

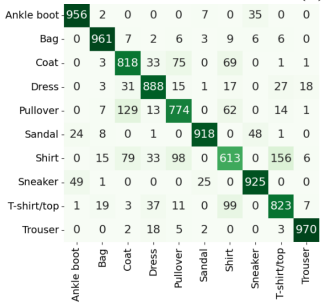
A 10-fold cross validation technique is then applied to each of these FFNN models to compare each optimisation algorithm with one another. Table VII presents the average and standard deviation of the performance metrics of the 10-fold cross validation for each optimisation algorithm.



(a) SGD



(b) SCG



(c) LFOP

Fig. 17: Confusion matrix of the Fashion MNIST dataset for each optimization algorithm, with true labels on the y-axis and predicted labels on the x-axis.

TABLE VII: Performance Metrics of each Optimisation Algorithm Obtained from Approximating the Linear Function

Algorithm	Performance metrics			
	mean mse	*std mse*	*mean time*	*std time*
SGD	0.0013	0	211.63	65.69
SCG	0.0415	0.0504	4.42	0.31
LFOP	0.0108	0.0283	2.91	0.58

The MSE scores obtained from each fold by each FFNN model are presented in Figure 18.

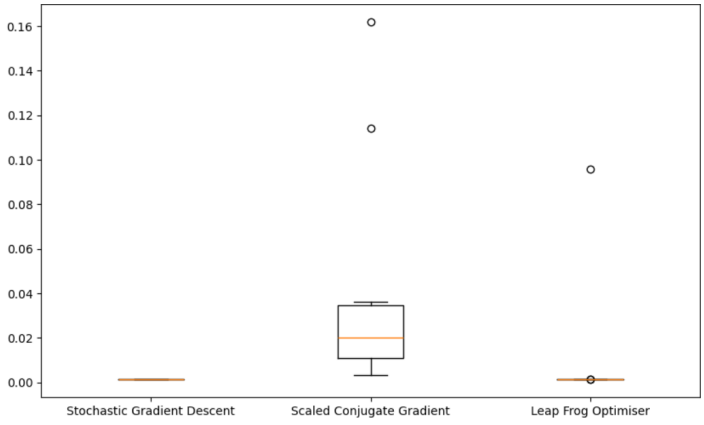


Fig. 18: Comparison of MSE for three optimisation algorithms when a linear function is approximated by use of 10-fold cross-validation.

The Kruskal-Wallis statistical test returned a p-value of 2.48×10^{-6}, which shows a significant difference between the scores of each optimisation algorithm. Figure 19 presents the average time each model takes for construction across the 10-folds.

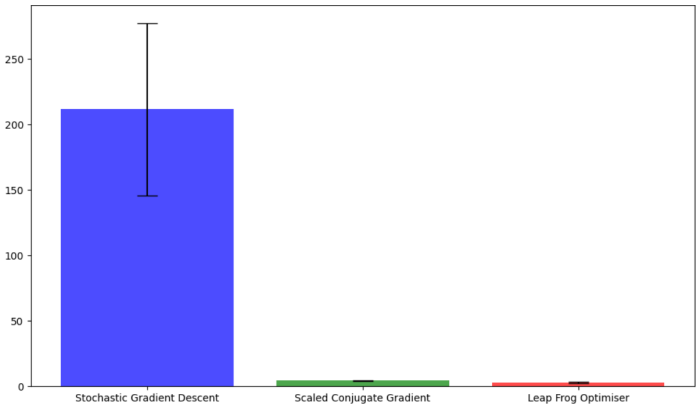
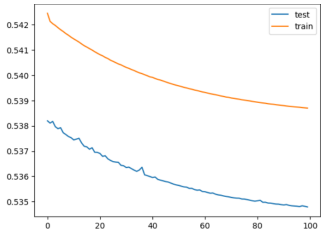
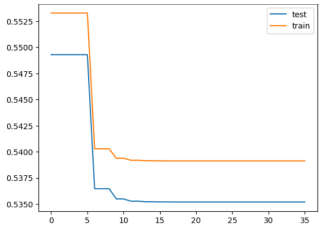


Fig. 19: Comparison of construction time for three optimisation algorithms when a linear function is approximated by use of 10-fold cross-validation.

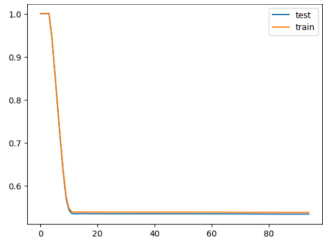
The models are then constructed on the same generated data. Figure 20 represents the training and validation errors over epochs for each optimization algorithm on this generated data.



(a) SGD



(b) SCG

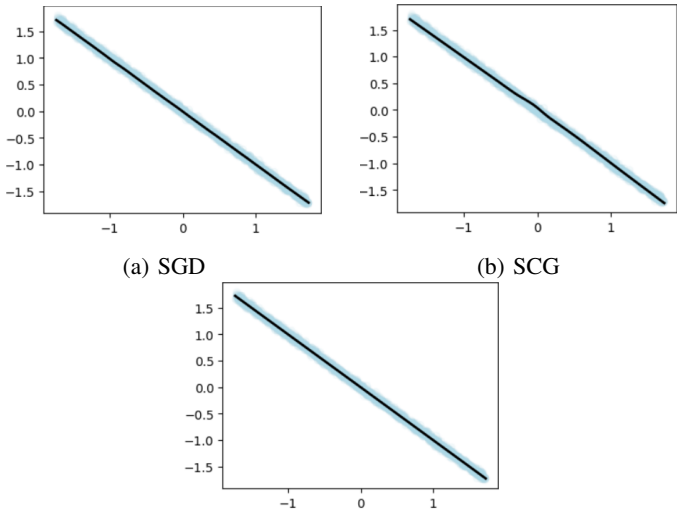


(c) LFOP

Fig. 20: Training and validation errors over epochs for the linear approximation task of each optimisation algorithm

All three the optimisation algorithms does not overfit to the data, as seen in Figure 28.

From this generated data, the output of each optimisation algorithm plots against the data to show how well the FFNN model approximates the linear function. These plots is presented in Figure 21.



(a) SGD (b) SCG

(c) LFOP

Fig. 21: Approximation of the linear function of each optimisation algorithm

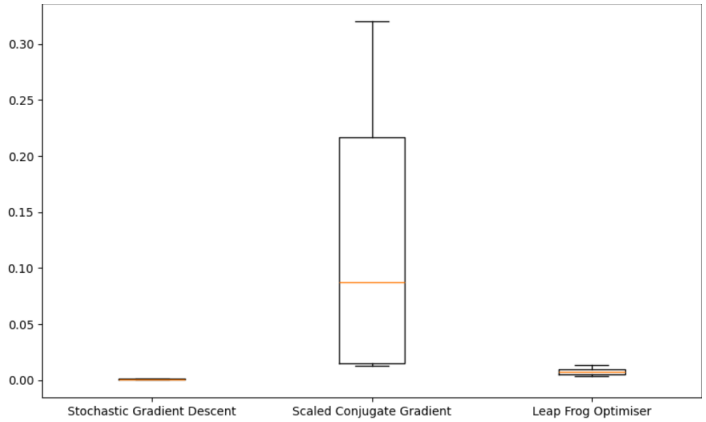


Fig. 22: Comparison of MSE for three optimisation algorithms when a cubic function is approximated by use of 10-fold cross-validation.

The SGD optimisation algorithm returns the lowest mean score across the 10-folds and has a standard deviation of zero. Additionally, the SGD algorithm approximates the linear function the best. Although this algorithm takes the longest to construct the FFNN model, the SGD is the optimal optimisation algorithm to use when the linear function is approximated.

The Kruskal-Wallis statistical test returned a p-value of 3.21×10^{-6}, which shows a significant difference between the scores of each optimisation algorithm. Figure 23 presents the average time each model takes for construction across the 10-folds.

E. Cubic Function Approximation

By use of Bayesian optimisation, the optimal values of the control parameters in the SGD, SGD, and LFOP optimisation algorithm are determined as seen in Section IV-D.

A 10-fold cross validation technique is then applied to each of these FFNN models to compare each optimisation algorithm with one another. Table VIII presents the average and standard deviation of the performance metrics of the 10-fold cross validation for each optimisation algorithm.

TABLE VIII: Performance Metrics of each Optimisation Algorithm Obtained from Approximating the Cubic Function

Algorithm	**Performance metrics**			
	mean mse	***std mse***	***mean time***	***std time***
SGD	0.0009	0.0003	270.96	5.39
SCG	0.1239	0.1190	18.46	2.49
LFOP	0.0080	0.0031	26.96	0.93

The MSE scores obtained from each fold by each FFNN model are presented in Figure 22.

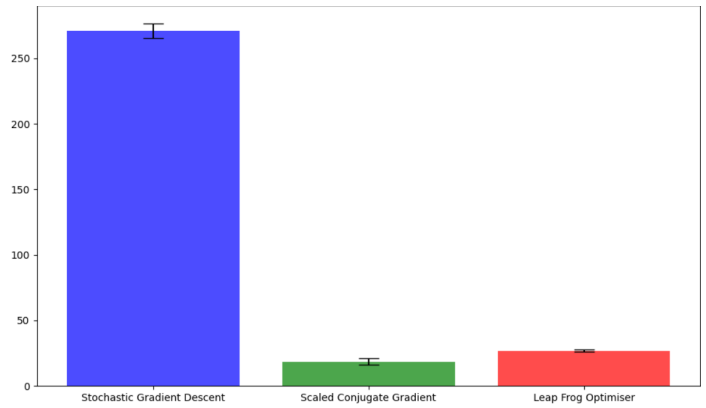
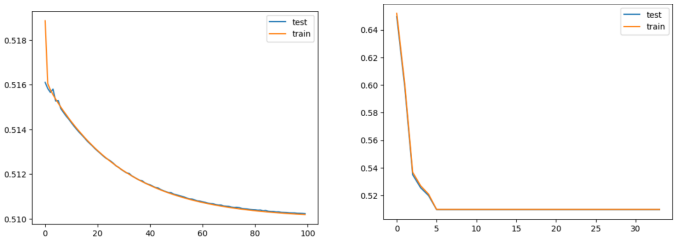
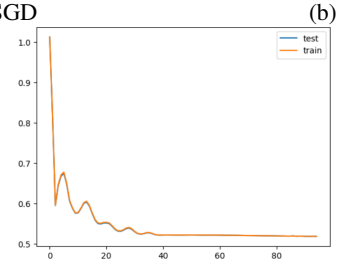


Fig. 23: Comparison of construction time for three optimisation algorithms when a cubic function is approximated by use of 10-fold cross-validation.

The models are then constructed on the same generated data. Figure 24 represents the training and validation errors over epochs for each optimization algorithm on this generated data.



(a) SGD (b) SCG

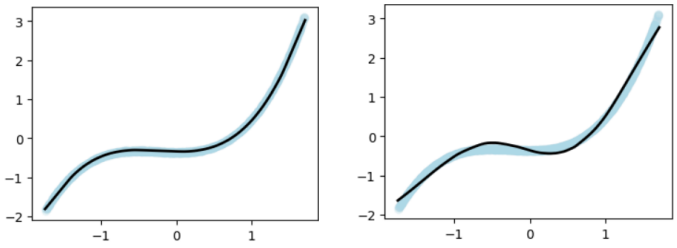


(c) LFOP

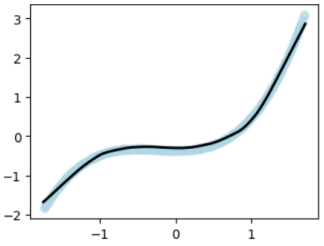
Fig. 24: Training and validation errors over epochs for the cubic approximation task of each optimisation algorithm

All three the optimisation algorithms does not overfit to the data, as seen in Figure 24.

From this generated data, the output of each optimisation algorithm plots against the data to show how well the FFNN model approximates the cubic function. These plots is presented in Figure 25.



(a) SGD (b) SCG



(c) LFOP

Fig. 25: Approximation of the cubic function of each optimisation algorithm

The SGD optimisation algorithm returns the lowest mean score across the 10-folds and has the lowest deviation between the scores. Additionally, the SGD algorithm approximates the cubic function the best. Although this algorithm takes the longest to construct the FFNN model, the SGD is the optimal optimisation algorithm to use when the cubic function is approximated.

F. Logistic Function Approximation

By use of Bayesian optimisation, the optimal values of the control parameters in the SGD, SGD, and LFOP optimisation algorithm are determined as seen in Section IV-D.

A 10-fold cross validation technique is then applied to each of these FFNN models to compare each optimisation algorithm with one another. Table IX presents the average and standard deviation of the performance metrics of the 10-fold cross validation for each optimisation algorithm.

TABLE IX: Performance Metrics of each Optimisation Algorithm Obtained from Approximating the Logistic Function

Algorithm	**Performance metrics**			
	mean mse	***std mse***	***mean time***	***std time***
SGD	0.064	0.0646	221.71	73.15
SCG	0.5036	0.2195	18.46	7.90
LFOP	0.1031	0.0137	119.14	3.45

The MSE scores obtained from each fold by each FFNN model are presented in Figure 26.

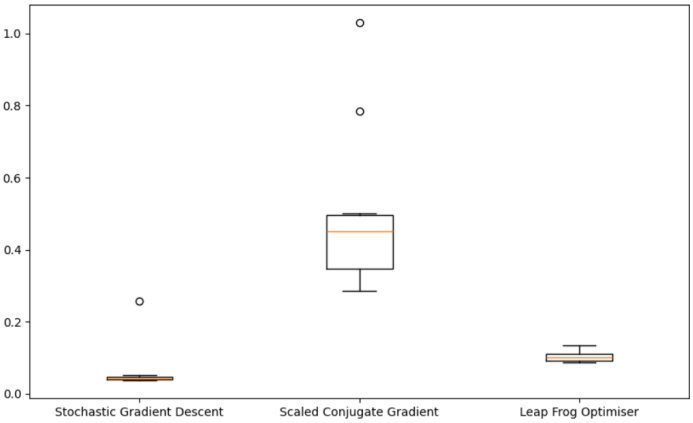


Fig. 26: Comparison of MSE for three optimisation algorithms when a logistic function is approximated by use of 10-fold cross-validation.

The Kruskal-Wallis statistical test returned a p-value of 7.95×10^{-6}, which shows a significant difference between the scores of each optimisation algorithm. Figure 27 presents the average time takes model takes for construction across the 10-folds.

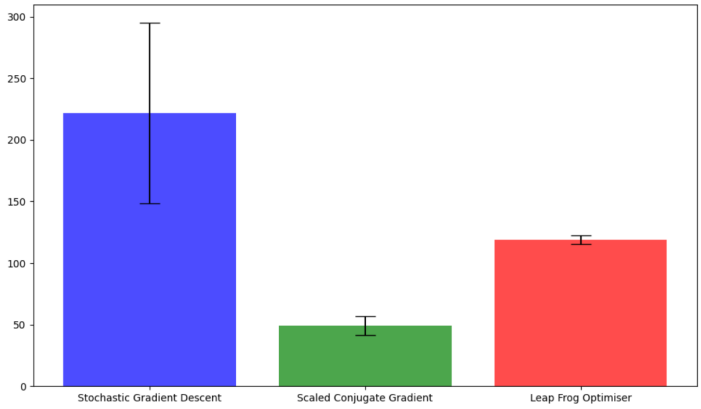
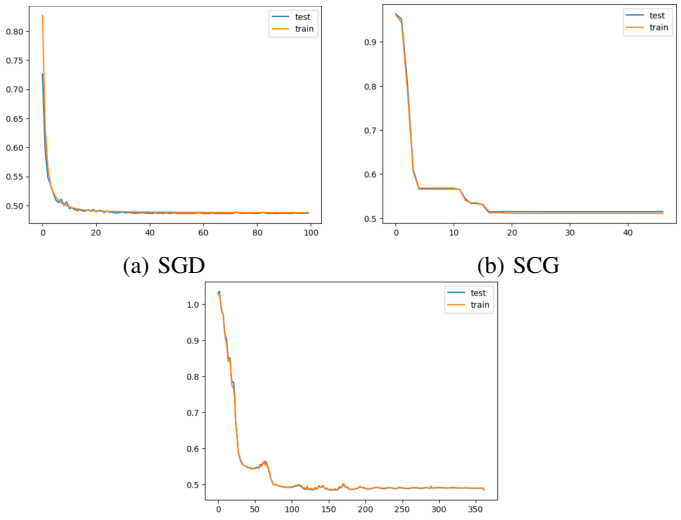


Fig. 27: Comparison of construction time for three optimisation algorithms when a logistic function is approximated by use of 10-fold cross-validation.

The models are then constructed on the same generated data. Figure 28 represents the training and validation errors over epochs for each optimization algorithm on this generated data.



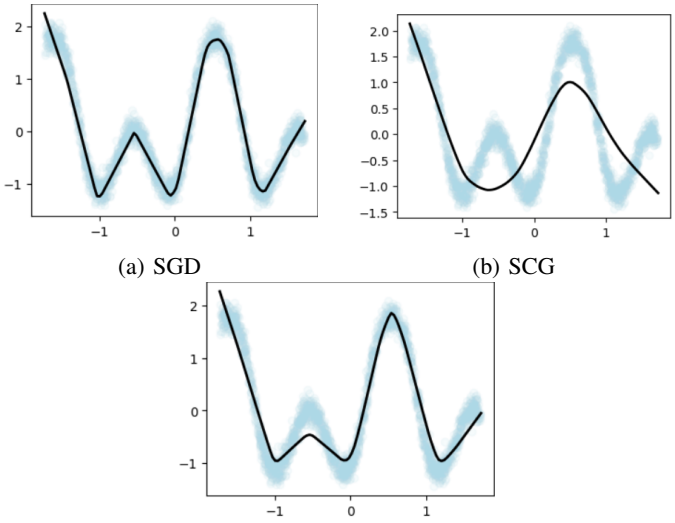
(a) SGD (b) SCG

(c) LFOP

Fig. 28: Training and validation errors over epochs for the logistic approximation task of each optimisation algorithm

All three the optimisation algorithms does not overfit to the data, as seen in Figure 28.

From this generated data, the output of each optimisation algorithm plots against the data to show how well the FFNN model approximates the logistic function. These plots is presented in Figure 29.



(a) SGD (b) SCG

(c) LFOP

Fig. 29: Approximation of the logistic function of each optimisation algorithm

The SGD optimisation algorithm returns the lowest mean score across the 10-folds and has the lowest deviation between the scores. Additionally, the SGD algorithm approximates the logistic function the best. Although this algorithm takes the longest to construct the FFNN model and there is one fold that generated a much higher MSE, the SGD is the optimal optimisation algorithm to use when the logistic function is approximated.

VI. Conclusion

This paper provides a comparative analysis of three optimisation algorithms used to adjust the weights of a feed forward neural network (FFNN) model. The three optimisation algorithms examined in this paper is the stochastic gradient descent (SGD) optimisation algorithm, the scaled conjugate cradient (SCG) optimisation algorithm, and the leap-frog optimisation (LFOP) algorithm. The results from three classification tasks and three function approximation tasks that vary in complexity reveal that each optimisation algorithm has distinct strenghts and weaknesses.

The SGD optimisation algorithm performs well in function approximation tasks, as it achieves the best performance on linear, cubic, and logistic function approximation tasks, although it is computationally expensive. The SGD optimisation algorithm performs the best on the fashion MNIST dataset, as the algorithm is able to handle complex data better than the rest of the optimisation algorithms. The SCG optimisation algorithm is quick to converge and generally works better on smaller datasets, as the optimisation algorithm performed the best on the breast cancer dataset. The LFOP algorithm performs the best on the large datasets, as it is less susceptible to local minima and ,therefore, tends to converge faster than the other optimisation algorithm. LFOP performed the best on the moderatly, which is the body performance.

REFERENCES

[1] A. Athaiya, S. Sharma, S. Sharma "Activation functions in neural networks." In: Towards Data Sci (2017).

[2] A. C. Bovik, Z. Wang "Mean squared error: Love it or leave it? A new look at signal fidelity measures." In: IEEE signal processing magazine (2009).

[3] Y. Bengio, L. Bottou, P. Haffner, Y. LeCun "Gradient-based learning applied to document recognition." In: Proceedings of the IEEE (1998).

[4] C. Ferri, J. Hernández-Orallo, R. Modroiu "An experimental comparison of performance measures for classification." In: Pattern recognition letters (2009).

[5] J. Hertz, A. Krogh "A simple weight decay can improve generalization." In: Advances in neural information processing systems (1991).

[6] G. E. Hinton, D. E. Rumelhart, R. J Williams. "Learning representations by back-propagating errors." (1986).

[7] G. E. Hinton, V. Nair "Rectified linear units improve restricted boltzmann machines." In: In Proceedings of the 27th international conference on machine learning (2010).

[8] J. Kiefer, J. Wolfowitz "Stochastic estimation of the maximum of a regression function." In: The Annals of Mathematical Statistics (1952).

[9] W. S. McCulloch, W. Pitts "A logical calculus of the ideas immanent in nervous activity." In: The bulletin of mathematical biophysics (1943).

[10] M. F. Møller "A scaled conjugate gradient algorithm for fast supervised learning." In: Neural networks (1993).

[11] H. Robbins, S. Monro "A stochastic approximation method." In: The annals of mathematical statistics (1951).

[12] F. Rosenblatt "The perceptron: a probabilistic model for information storage and organization in the brain." In: Psychological review (1958).

[13] J. A. Snyman "A new and dynamic method for unconstrained minimization." In: Applied Mathematical Modelling (1982).