**ENHANCED KNOWLEDGE IN SCIENCES AND TECHNOLOGY**
e-ISSN: 2773-6385

**EKST**

Vol. 4 No. 2 (2024) 171-180
https://publisher.uthm.edu.my/periodicals/index.php/ekst

# Performance of Different Activation Functions in Two Hidden Layer Handwritten Digit Recognition Neural Network

**Leong Yew Joe[1], Lee Siaw Chong[1*]**

[1] *Department of Mathematics and Statistics, Faculty of Applied Sciences and Technology, UTHM Kampus Cawangan Pagoh, Hab Pendidikan Tinggi Pagoh, KM 1, Jalan Panchor, 84600 Pagoh, Muar, Johor, MALAYSIA*

*Corresponding Author: sclee@uthm.edu.my
DOI: https://doi.org/10.30880/ekst.2024.04.02.021

**Article Info**

Received: 27 December 2023
Accepted: 11 January 2024
Available online: 12 December 2024

**Keywords**

Neural Network, Handwritten Digit Recognition, Activation Function, Hyperbolic Tangent Function, Sigmoid Function, Rectified Linear Unit Function

**Abstract**

This research investigates on handwritten digit recognition using neural networks, with the primary objective of comparing the performance of three distinct activation functions—tanh, sigmoid, and ReLU. Initial investigations focus on identifying optimal parameters involving the weight initialization method, the number of nodes in hidden layers, and learning rate. Through systematic experimentation, the study reveals the most effective configurations for these parameters within certain limitations. Subsequently, the chosen configurations are employed to evaluate the performance of the neural network for each activation function. Results indicate that the ReLU function outperforms both sigmoid and tanh functions, establishing itself as the most effective activation function for the given task provided that the optimal parameters are found. These findings provide valuable insights for optimizing neural network architectures in handwritten digit recognition applications.

## 1. Introduction

Handwritten digit recognition (HDR) is an extensively explored area within the field that is concerned with learning models for distinguishing pre-segmented handwritten digits [1]. HDR can also be defined as the task of identifying and classifying handwritten digits using machines automatically. The goal is to create algorithms and models that are capable of categorizing and interpreting these handwritten digits into their corresponding class labels effectively [2]. It is usually focused on identifying the digits ranging from 0 to 9 since the number system base 10 is the most common number system that is used in the real world.

HDR is found to be used in various applications such as postal mail sorting, automated form processing, digitalizing old documents, etc. According to Srihari, Shekhawat, and Lam, HDR is a subset of optical character recognition (OCR), which is defined as the electronic process of converting images of handwritten numerals, letters, and symbols from scanned documents and photos into a computer-processable format [3]. By converting physical text into digital format, productivity and accessibility in a variety of industries can be increased, enabling effective text analysis, indexing, and manipulation.

As we enter the fourth revolution, artificial intelligence (AI) has received a lot of attention in various fields of study. While the definition of AI is ambiguous, it can be defined as the study of how to build or program computers to enable them to do what minds can do [4]. The broad term AI includes machine learning (ML), deep learning, and artificial neural networks (ANN).

The term ML can be referred to as a group of procedures or techniques that have the ability to 'learn' from input data and use the inherent historical patterns in the data to compute and make predictions or decisions without being explicitly programmed [5]. Deep learning and ANN are subsets and fall under the branch of ML. They incorporate with one another to create deep complex neural networks, which are used in designing models used for complicated problems.

There are many ways to create an HDR model, such as statistical techniques, structural techniques, neural network approach, template matching, fuzzy model, and hybrid model. The neural network approach is the most popular technique among all the others. ANN is a computing system whose main idea is modelled after biological neural networks [6]. It imitates how the brain sends signals from one neuron to another.

ANN is made up of nodes, connections, and layers, where each node in a layer is fully connected to the nodes in the next layer [7]. Each connection represents a computation function whereby it receives inputs from the previous layer, performs a computation, and returns outputs to the next layer. One of the basic computations in the nodes is the activation function, which plays a crucial role when designing the ANN [8].

The selection of activation function is an important consideration in the design of neural networks for digit recognition. Non-linearities are introduced into the network by activation functions, allowing it to capture complex relationships between the inputs and outputs. This allows the networks to learn from data and adapt, allowing them to recognise complex patterns and make precise predictions. Different activation functions have distinctive properties that can influence the performance of the neural network for digit recognition.

The aim of this research is to build a Python-based handwritten digit recognition artificial neural network without built-in libraries. Although Python has a number of libraries for machine learning, including TensorFlow and Keras, this project does not use them. The neural network is then used to determine the best parameters to be used for each activation function from the weight initialization method, followed by the number of nodes in the hidden layers and the learning rate. These parameters are then used to evaluate the performance of sigmoid, tanh, and ReLU activation functions in the constructed neural network by analysing their error rates, training accuracy and testing accuracy.

## 2. Methodology

This project can be divided into four phases, which are data acquisition, designing the model, training the model, and evaluating the model. Data acquisition is first made to obtain the handwritten digit data that is used in the project. Secondly, handwritten digit recognition (HDR) model is designed and built in Python. Thirdly, the data obtained is used to train the HDR model. Finally, the model is evaluated to compare the performance of different activation functions.

The data used in the project is publicly available from the MNIST database. The data includes 42,000 black and white handwritten digits with labels. The data is saved in a comma-separated values (CSV) file. Each digit is represented by 28 by 28 pixels. Each pixel holds a value between 0 and 255, where a higher value represents a higher intensity of the black pixel. The first column of the dataset is the label of the digit, while the remaining columns correspond to the 784 pixels. Figure 1 shows an example of the dataset in Microsoft Excel.

| | A | HU | HV | HW | HX | HY | HZ | IA | IB | IC | ID | IE | IF | IG | IH |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | label | pixel227 | pixel228 | pixel229 | pixel230 | pixel231 | pixel232 | pixel233 | pixel234 | pixel235 | pixel236 | pixel237 | pixel238 | pixel239 | pixel240 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 29 |
| 3 | 0 | 0 | 0 | 0 | 61 | 191 | 254 | 254 | 254 | 254 | 254 | 109 | 83 | 199 | 254 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 254 | 254 | 184 | 0 |
| 5 | 4 | 0 | 0 | 0 | 0 | 160 | 207 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 23 | 210 | 253 | 253 | 253 | 248 | 161 | 222 | 222 | 246 | 253 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 211 | 254 | 58 | 0 | 0 | 0 | 0 |
| 8 | 7 | 0 | 0 | 0 | 122 | 253 | 252 | 253 | 252 | 223 | 243 | 253 | 252 | 253 | 252 |
| 9 | 3 | 0 | 0 | 0 | 207 | 254 | 254 | 177 | 117 | 39 | 0 | 0 | 56 | 248 | 102 |
| 10 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 197 | 254 | 253 | 165 | 2 | 0 | 0 |
| 11 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 28 | 206 | 253 |
| 12 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 38 | 155 | 252 | 252 | 252 | 253 | 252 |
| 13 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 166 | 241 | 252 | 253 | 252 |
| 14 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 89 | 254 | 254 | 105 |
| 15 | 3 | 128 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 | 255 |
| 16 | 3 | 0 | 0 | 0 | 57 | 252 | 173 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 25 |
| 17 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 92 | 253 |

**Fig. 1** Example of the dataset in Excel

Besides representing the data in the form of rows and columns, the dataset can also be visualized as images with 28 by 28 pixels. Fig. 2 shows the example of the first 20 data from the MNIST dataset in the form of images.

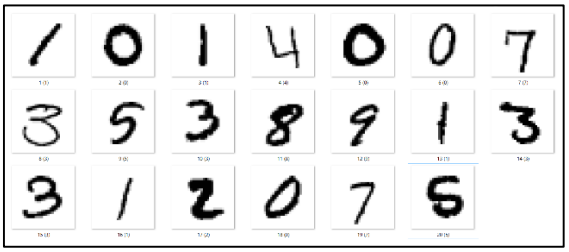


**Fig. 2** Example of the first 20 data in the form of images

Enhanced Knowledge in Sciences and Technology Vol. 4 No. 2 (2024) p. 171-180

The dataset does not have to be cleaned as it has already been pre-processed. The dataset will be separated into a training set and a testing set. The training set is used to train the neural network model, while the test set is used to evaluate the final neural network model. The data are split into a ratio of 70/30 for the training set and the testing set, respectively.

The artificial neural network (ANN) is a network of interconnected nodes, also known as artificial neurons or perceptrons, organized in layers. ANN consists of three layers which are the input layer, the hidden layer, and the output layer. Each layer consists of nodes where each node processes the input received using the activation function and outputs them with a value. Each connection is assigned a weight, and each node apart from the nodes from the input layer is assigned a bias.

The initialization of weight can be done by using random normal distribution. However better weight initialization methods such as Xavier initialization and He/Kaiming initialization are also widely used to prevent vanishing and exploding gradient problems. The Xavier initialization sets the initial weights by drawing them from a uniform probability distribution between the range $-1/\sqrt{n}$ and $1/\sqrt{n}$, where n is the size of the previous layer [9]. Meanwhile, He initialization sets the initial weight by drawing them from a Gaussian probability distribution with a mean of 0 and a standard deviation of $\sqrt{2/n}$, where n is the size of the previous layer [10].

For this project, the number of nodes at the input layer is 784, which corresponds to the number of pixels of each digit image in the dataset. The output layer consists of 10 nodes, which correspond to the digits 0 to 9. The number of hidden layers is set to 2, and the number of nodes is set to 30 and 15 for the first and second hidden layers, respectively.

The activation function is one of the components of designing a successful neural network [11]. There are many activation functions that can be used. In this project, three activation functions are tested and compared based on their performance. The activation functions are:

•Sigmoid/Logistic Function
•Hyperbolic Tangent Function (tanh)
•Rectified Linear Unit Function (ReLU)

The sigmoid/logistic function is a function that accepts any real values as its input and returns the output in a range of 0 to 1. As the input value tends to be positive infinity, the closer the output value will be to 1, while the input value tends to be negative infinity, the closer the output value will be to 0. The graph of the sigmoid/logistic function is shown in Fig. 3. The formula of the sigmoid/logistic function is:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

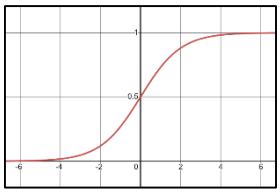


**Fig. 3** Sigmoid/Logistic Graph

The tanh function is defined as the ratio between the hyperbolic sine and hyperbolic cosine functions. The tanh function is similar to the sigmoid function as both have the same S-shape graph with a difference in an output range of $-1$ to 1. For the tanh function, the larger the input value, the closer the output value will be to 1, while the smaller the input value, the closer the output value will be to $-1$. The graph of the hyperbolic tangent graph is shown in Fig. 4. The formula of the tanh function is:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2}$$

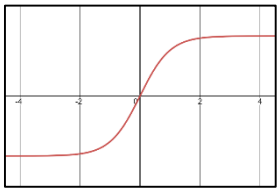


**Fig. 4** Hyperbolic Tangent Graph

The rectified linear unit (ReLU) function is an activation function that returns 0 for all negative input values but for any positive input value, it returns the same value. Although the ReLU function gives the impression of a linear function, it has a derivative function and allows backpropagation. The graph of the ReLU function is shown in Fig. 5. The formula of the ReLU function is:

$$f(x) = \max(0, x) \tag{3}$$

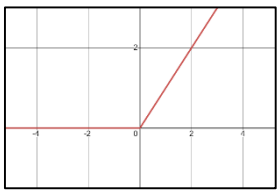


**Fig. 5** Rectified Linear Unit (ReLU) Graph

The training of the neural network model is the process of adjusting the weight and biases of the network to optimize the performance of recognizing the handwritten digits. The training of the neural network can be divided into two phases which are forward propagation and backward propagation.

Forward propagation, which is also known as feedforward propagation, is the process of passing input data through a neural network to produce an output or prediction [12]. The input data is passed through the neural network's layers in a forward direction, from the input layer to the output layer.

Each neuron in the network receives inputs from the previous layer, performs a weighted sum of these inputs, applies an activation function, and produces an output during the process. This output is then passed to the neurons in the next layer as its input, and the process is repeated until the output layer is reached. The formula for the weight sum is as follows:

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \tag{4}$$

$W^{(l)}$ = weights matrix for the $l^{\text{th}}$ layer,
$\mathbf{b}^{(l)}$ = bias vector for the $l^{\text{th}}$ layer,
$\mathbf{z}^{(l)}$ = weight sum vector of $l^{\text{th}}$ layer,
$\mathbf{a}^{(l-1)}$ = output vector of the $(l-1)^{\text{th}}$ layer.

After the weight sum is found for one layer, the activation function is applied to the weight sum. The formula is as follows:

$$\mathbf{a}^{(l)} = f\left(\mathbf{z}^{(l)}\right), \tag{5}$$

$\mathbf{z}^{(l)}$ = weight sum vector of $l^{\text{th}}$ layer,
$\mathbf{a}^{(l)}$ = output vector of $l^{\text{th}}$ layer.

At the output layer, the softmax function is applied to convert the vector into a vector that sums to 1. In other words, the softmax function transforms the input values into values between 0 to 1, and can be interpreted as a probability. The formula of the softmax function is as follows. Since the output has 10 elements which represent the digits 0 to 9, the denominator takes the summation of the exponent for all 10 weight sum of the output.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}, \tag{6}$$

$\mathbf{z}$ = weight sum vector at the output layer,
$\sigma$ = softmax function.

Finally, the mean square error (*MSE*) is used as the cost function of the neural network model. It calculates the average squared difference between the predicted output and the target output. The formula for *MSE* is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\text{a}_i - \text{y}_i)^2, \tag{7}$$

$n$ = number of samples in the dataset,

Enhanced Knowledge in Sciences and Technology Vol. 4 No. 2 (2024) p. 171-180

$a_i$ = $i^{th}$ element in the output vector of the output layer,
$y_i$ = $i^{th}$ element in the vector of the target value.

Backward propagation, otherwise known as backpropagation, is the process of updating the weights and biases by propagating the error layer by layer backward through the neural network. The error is used to determine the gradient of the cost function which indicates the adjustment needed to be made to the weights and biases in minimizing the error.

The calculation of the gradient of the cost function involves the calculation of the error gradient first. The error gradient evaluates how sensitive the cost function is with respect to the inputs of a specific node. It is computed as the derivative of the activation function with respect to the weighted input of the node. The formula is as follows:

$$\frac{dC}{d\mathbf{a}} = \boldsymbol{\delta}^{(l)} = \left(W^{(l+1)}\boldsymbol{\delta}^{(l+1)}\right) \odot f'\left(\mathbf{z}^{(l)}\right), \tag{8}$$

$\boldsymbol{\delta}^{(l)}$ = error gradient vector at the $l^{\text{th}}$ layer,
$W^{(l+1)}$ = weight matrix at the $(l+1)^{\text{th}}$ layer,
$\boldsymbol{\delta}^{(l+1)}$ = error gradient vector at the $(l+1)^{\text{th}}$ layer,
$f'$ = derivative of the activation function,
$\mathbf{z}^{(l)}$ = weight sum vector at the $l^{\text{th}}$ layer.

The error gradients are then used to compute the gradients of the cost function with respect to each weight and bias. These gradients are then used to update the weights and biases. The formulas are as follows:

$$\Delta\mathbf{w}^{(l)} = \alpha\mathbf{a}^{(l-1)}\boldsymbol{\delta}^{(l)}, \tag{9}$$
$$\Delta\mathbf{b}^{(l)} = \alpha\boldsymbol{\delta}^{(l)}, \tag{10}$$

$\Delta\mathbf{w}^{(l)}$ = gradient of the cost function with respect to the weight vector,
$\Delta\mathbf{b}^{(l)}$ = gradient of the cost function with respect to the bias vector,
$\alpha$ = learning rate,
$\mathbf{a}^{(l-1)}$ = output matrix of the $(l-1)^{\text{th}}$ layer,
$\boldsymbol{\delta}^{(l)}$ = error gradient at the $l^{\text{th}}$ layer.

After completing the training of the HDR neural network model, the model is evaluated by feeding the test set into the model. The model will predict the test set, and the prediction outcomes are in the form of class labels. The prediction is then compared to the true labels of the digits. The performance of the model can be evaluated by calculating the accuracy of the prediction where accuracy is the proportion of correctly classified digits out of the total number of digits in the test set.

## 3. Result and Discussion

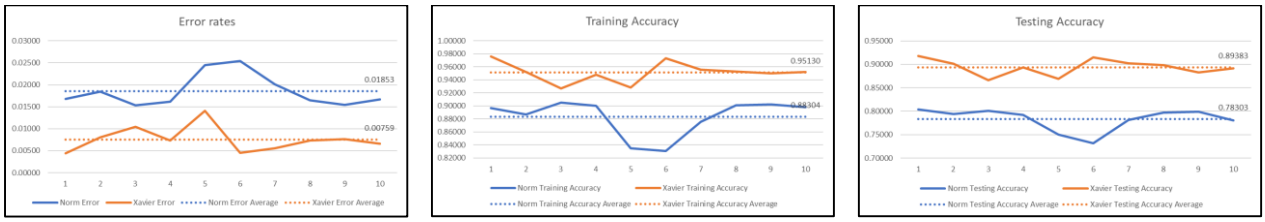
The research is conducted by first determining the optimal parameters for the neural network models associated with each activation function — tanh, sigmoid, and ReLU. The primary focus lies in identifying the most effective weight initialization method followed by the number of nodes in the hidden layer and the learning rate. The optimal parameters identified are used in the subsequent investigations. In the end, with the confirmed optimal parameters in place for each activation function, these parameters are used when configuring the models for a comprehensive comparison of the performance of each activation function at their peak efficiency.

### 3.1 Weight Initialization

It is proposed that Xavier initialization works well with tanh and sigmoid activation functions [9]. Meanwhile, He initialization works well with the ReLU activation function [10]. During weight initialization, the use of normal distribution and Xavier initialization are compared for both tanh and sigmoid activation functions. As for the ReLU function, the use of normal distribution and He initialization are compared. The performance of the neural network model is evaluated by taking an average from 10 sample models for each case. Other parameters involved in setting the neural network models are fixed as in Table 1:

**Table 1** ***Parameters fixed during weight initialization comparison***

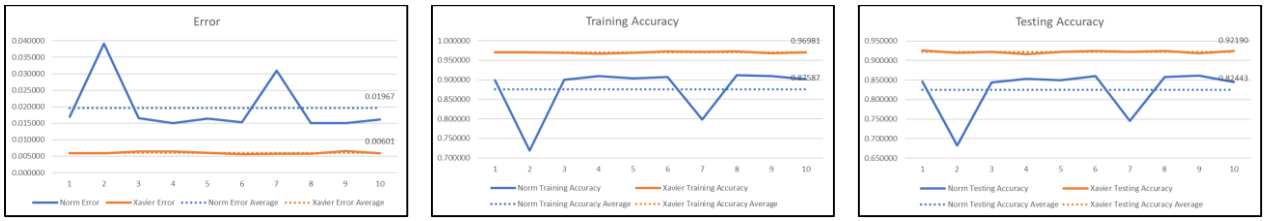
| Parameters | Value |
|---|---|
| Number of nodes in the first hidden layer, $h_1$ | 30 |
| Number of nodes in the second hidden layer, $h_2$ | 15 |
| Number of epochs | 30 |
| Learning rate | 0.1 |
| Number of samples in training set | 7000 |
| Number of samples in testing set | 3000 |




**(a)** **(b)** **(c)**

**Fig. 6** (a) error rate; (b) training accuracy; (c) testing accuracy of normal distribution and Xavier initialization methods for tanh activation function

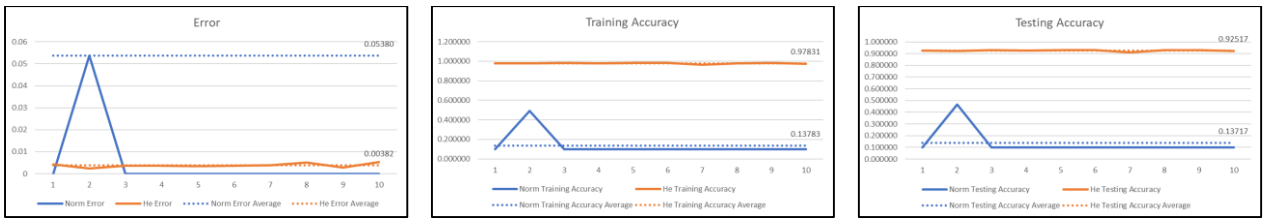
Fig. 6 shows the performance of normal distribution and Xavier initialization methods for tanh activation function in terms of error rate, training accuracy and testing accuracy. Each graph shows the values of the performance of the 10 sample models. The average of each performance is indicated by the dotted line in the graphs.




**(a)** **(b)** **(c)**

**Fig. 7** (a) error rate; (b) training accuracy; (c) testing accuracy of normal distribution and Xavier initialization methods for sigmoid activation function

Fig. 7 shows the performance of normal distribution and Xavier initialization methods for sigmoid activation function in terms of error rate, training accuracy and testing accuracy. Each graph shows the values of the performance of the 10 sample models. The average of each performance is indicated by the dotted line in the graphs.




**(a)** **(b)** **(c)**

**Fig. 8** *(a) error rate; (b) training accuracy; (c) testing accuracy of normal distribution and He initialization methods for ReLU activation function*

Fig. 8 shows the performance of normal distribution and He initialization methods for ReLU activation function in terms of error rate, training accuracy and testing accuracy. Each graph shows the values of the performance of the 10 sample models. The average of each performance is indicated by the dotted line in the graphs.

Enhanced Knowledge in Sciences and Technology Vol. 4 No. 2 (2024) p. 171-180

**Table 2** Performance of activation functions with different weight initialization methods

| Activation Function | Weight initialization method | Average error rates | Average training accuracy (%) | Average testing accuracy (%) |
|---|---|---|---|---|
| Tanh | Normal Distribution | 0.01853 | 88.30 | 78.30 |
| | Xavier Initialization | 0.00759 | 95.13 | 89.38 |
| Sigmoid | Normal Distribution | 0.01967 | 87.59 | 82.44 |
| | Xavier Initialization | 0.00601 | 96.98 | 92.19 |
| ReLU | Normal Distribution | 0.05380 | 13.78 | 13.71 |
| | He Initialization | 0.00382 | 97.83 | 92.52 |

From Table 2, it is shown that using Xavier and He weight initialization methods outperform those that used normal distribution in terms of average error rates, average training accuracy and average testing accuracy for all three activation functions. A proper weight initialization method provides a better starting point for the model to learn.

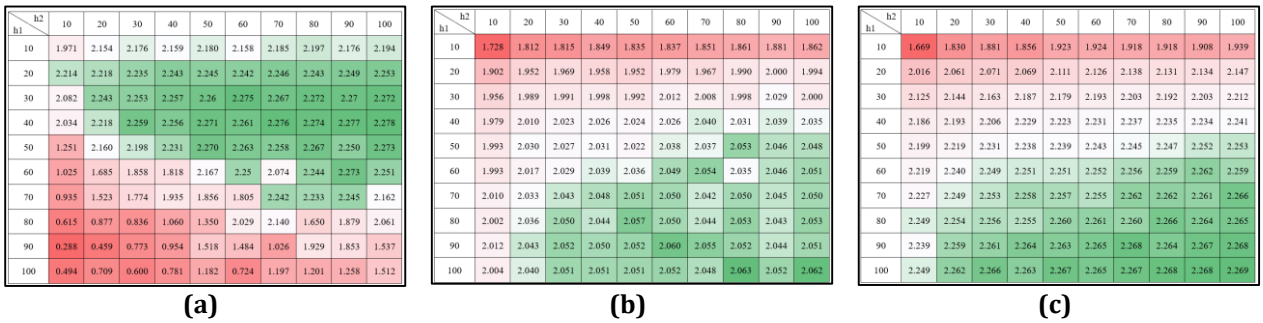
## 3.2 Number of Nodes in Hidden Layers

In this section, we investigate how varying the number of nodes in both hidden layers within the range of 10 to 100 with increments of 10 affects the performance of the models. For each configuration, 10 sample models are trained, and the performances are evaluated by taking the average from the 10 samples. The weight initialization methods used are Xavier initialization for tanh and sigmoid functions and He initialization for the ReLU function. The number of epochs is set to 30 while the learning rate is set to 0.1. The number of samples in the training and the testing set are 7000 and 3000, respectively.

The performance of the models is evaluated based on the error rates, accuracy of the training set and accuracy of the testing. Certain weightages decided beforehand are used to combine the performances into an index. The index serves as a measure of the overall performance of each model. The average performance for each configuration is taken from 10 sample models. The corresponding weightage values are outlined in Table 3:

**Table 3** Weightage for the performance of the model

| Performance of model | Weightage |
|---|---|
| Error rates | 0.5 |
| Accuracy of training set | 0.8 |
| Accuracy of testing set | 1.0 |




**(a)** **(b)** **(c)**

**Fig. 9** Overall performance for varying numbers of nodes for functions of (a) tanh; (b) sigmoid; (c) ReLU

Fig. 9 shows the overall performance for each activation function with varying numbers of nodes for the first and second hidden layers. From Fig. 9 (a), it is shown that the tanh model performs the best with 40 nodes in the first hidden layer and 100 nodes for the second hidden layer with an overall performance index of 2.278. Meanwhile, Fig. 9 (b) shows that the top performing sigmoid models are concentrated at the bottom of the table. The sigmoid model performs the best with 100 nodes in the first hidden layer and 80 nodes in the second hidden layer with an overall performance index of 2.063. Last but not least, Fig. 9 (c) shows that the model performs the best with 100 nodes in the first hidden layer and 100 nodes in the second hidden layer with an overall performance index of 2.269.

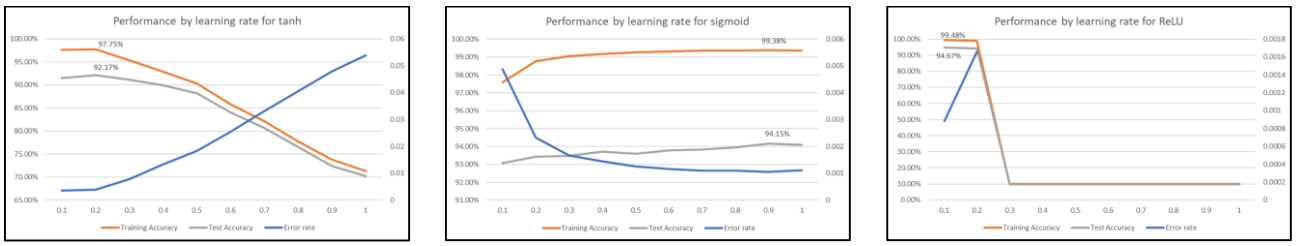
**Table 4** Activation function, overall performance and number of nodes

| Activation Function | Error rates | Accuracy of training set (%) | Accuracy of training set (%) | Overall performance | Number of nodes in first hidden layer, $h_1$ | Number of nodes in second hidden layer, $h_2$ |
|---|---|---|---|---|---|---|
| Tanh | 0.003627 | 97.68 | 91.65 | 2.278 | 40 | 100 |
| Sigmoid | 0.004792 | 97.61 | 92.91 | 2.063 | 100 | 80 |
| ReLU | 0.000785 | 99.53 | 94.47 | 2.269 | 100 | 100 |

Table 4 shows the overall performance and number of nodes for the best performing model among all different configuration of a number of nodes for each activation function. After training and evaluating each configuration of nodes for each activation function, tanh function performs the best when the number of nodes for the first and second hidden layers are 40 and 100, respectively. As for the sigmoid function, the model outstands when the number of nodes is 100 and 80, respectively. The ReLU activation function utilizes 100 nodes in both hidden layers to achieve the best result among all the other configurations.

## 3.3 Learning Rate

In this part, we tune the learning rate of the models from 0.1 to 1.0 with a step size of 0.1. The weight initialization method used is based on the result of the first investigation, and the number of nodes for both hidden layers for each activation function is based on the result from the second investigation. The remaining parameters, which are the number of training and testing sets, are fixed to be 7000 and 3000, respectively. The performance of the models is based on the error rates, accuracy of the training set and accuracy of the testing set. 10 sample models are trained for each learning rate, and the average is calculated for each performance.




**(a)** **(b)** **(c)**

**Fig. 10** Performance by learning rate (a) tanh; (b) sigmoid; (c) ReLU

Fig. 10 shows the performance of varying learning rates for each activation function. Fig. 10 (a) shows that the training and testing accuracies for the tanh activation function peak at 0.2 learning rate and decline as the learning rate increases. As for the sigmoid activation function shown in Fig. 10 (b), the training and testing accuracies increase from 0.1 learning rate and reach the highest values at 0.9 learning rate. Finally, Fig. 10 (c) shows that the 0.1 learning rate has the highest training and testing accuracies for the ReLU activation function.

**Table 5** Performance of activation function with the best learning rate

| Activation Function | Error rates | Accuracy of training set (%) | Accuracy of testing set (%) | Learning Rate |
|---|---|---|---|---|
| Tanh | 0.003937 | 97.75 | 92.17 | 0.2 |
| Sigmoid | 0.001047 | 99.38 | 94.15 | 0.9 |
| ReLU | 0.000882 | 99.48 | 94.67 | 0.1 |

Table 5 shows the performance of the best performing model among different learning rates for each activation function. After training and evaluating each activation function in the specified configuration with varying learning rates, the models demonstrated the best performance at 0.2 learning rate for the tanh function, 0.9 learning rate for the sigmoid function and 0.1 learning rate for the ReLU function. These learning rates are used in the next section for the comparison between each activation function.

Enhanced Knowledge in Sciences and Technology Vol. 4 No. 2 (2024) p. 171-180

## 3.4 Comparison between Tanh, Sigmoid, and ReLU functions

Table 6 concludes the optimal parameters for each activation function. These parameters are used in configurating the models for the comparison between tanh, sigmoid, and ReLU functions.

**Table 6** Parameters used for each activation function

| Activation Function | Tanh | Sigmoid | ReLU |
|---|---|---|---|
| Weight Initialization | Xavier Initialization | Xavier Initialization | He Initialization |
| Number of Nodes in First Hidden Layer, $h_1$ | 40 | 100 | 100 |
| Number of Nodes in Second Hidden Layer, $h_2$ | 100 | 80 | 100 |
| Learning rate | 0.2 | 0.9 | 0.1 |
| Number of epochs | 30 | 30 | 30 |
| Number of samples in training set | 16800 | 16800 | 16800 |
| Number of samples in testing set | 7200 | 7200 | 7200 |

**Table 7** Performance of each activation function

| Activation Function | Error rates | Training accuracy (%) | Testing accuracy (%) |
|---|---|---|---|
| Tanh | 0.00437 | 97.47 | 93.75 |
| Sigmoid | 0.00081 | 99.54 | 95.77 |
| ReLU | 0.00068 | 99.58 | 96.31 |

From Table 7, the ReLU activation function demonstrates the lowest error rate (0.00068) and superior accuracy in predicting both training and testing data. The sigmoid function follows closely with a very low error rate (0.00081), while the tanh function exhibits a somewhat higher error rate (0.00437).

In this specific configuration of parameters for each activation function, the ReLU function stands out with the highest training accuracy at 99.58%, indicating its strong ability to fit the training dataset. Remarkably, the ReLU function also exhibits the highest testing accuracy at 96.31%, showcasing its effectiveness in generalizing well to new, unseen data.

Sigmoid function follows closely with high training and testing accuracies of 99.54% and 95.77%, respectively. Tanh function, while demonstrating a respectable training accuracy of 97.47%, lags in testing accuracy at 93.75%. The consistency in performance metrics across training and testing phases is crucial for assessing a model's ability to not only memorize training data but also make accurate predictions on unfamiliar instances. These results suggest that all activation functions strike a good balance between generalization and overfitting with the ReLU function surpassing tanh and sigmoid functions in all performances.

## 4. Conclusion

As a conclusion, the architecture specification for a neural network plays an important role in the performance of the neural network model, which is demonstrated by using different weight initialization methods, varying the number of nodes in hidden layers and the learning rate. Our results demonstrated improved performance when employing the Xavier initialization method for tanh and sigmoid functions, and the He initialization method for the ReLU function. Results showed that choosing the right weight initialization method is crucial for the model to train in the first place. From our investigation, the tanh function performs the best, with 40 nodes in the first hidden layer and 100 nodes in the second hidden layer. The sigmoid function excels with 100 nodes in the first layer and 80 in the second, while ReLU function performs optimally with 100 nodes in both hidden layers. In our research, we also discovered that a learning rate of 0.2 yields the best performance for the tanh function, while the sigmoid function achieves optimal results with a learning rate of 0.9, and the ReLU function performs best with a learning rate of 0.1. Finally, the ReLU activation function emerges as the most effective among tanh and sigmoid activation functions in the specified specification of this research in the digit recognition task. These

findings underscore the importance of thoughtful architecture design and parameter selection in enhancing the neural network's ability to learn and generalize effectively.

## Acknowledgement

The authors would also like to thank the Faculty of Applied Sciences and Technology, Universiti Tun Hussein Onn Malaysia for its support.

## Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

## Author Contribution

*The authors confirm contribution to the paper as follows:* ***study conception and design****: Leong Yew Joe, Lee Siaw Chong;* ***data collection****: Leong Yew Joe, Lee Siaw Chong;* ***analysis and interpretation of results****: Leong Yew Joe, Lee Siaw Chong;* ***draft manuscript preparation****: Leong Yew Joe, Lee Siaw Chong. All authors reviewed the results and approved the final version of the manuscript.*

## References

[1] Shamim, S. M., Miah, M. B. A., Sarker, A., Rana, M., & Al Jobair, A. (2018). Handwritten digit recognition using machine learning algorithms. *Global Journal Of Computer Science And Technology*, *18*(1), 17-23.
[2] Kulkarni, S. R., & Rajendran, B. (2018). Spiking neural networks for handwritten digit recognition—Supervised learning and network optimization. *Neural Networks*, 103, 118-127.
[3] Srihari, S. N., Shekhawat, A., & Lam, S. W. (2003). Optical character recognition (OCR). In: Anthony R., Edwin D. R., and David H. *Encyclopedia of Computer Science*. United Kingdom: John Wiley and Sons Ltd. pp. 1326-1333;2003.
[4] Boden, M. A. (1996). *Artificial Intelligence*. Elsevier.
[5] Rebala, G., Ravi, A., & Churiwala, S. (2019). *Machine learning definition and basics.* Springer Cham.
[6] Dongare, A. D., Kharde, R. R., & Kachare, A. D. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1), 189-194.
[7] Aggarwal, C. C. (2018). *Neural networks and deep learning*. Springer Cham.
[8] Lek, S., & Park, Y. S. (2008) Artificial neural networks. In: Sven E. J. & Brian D. F. *Encyclopedia of Ecology*. Amsterdam, Netherlands: Elsevier Inc. pp. 237-245.
[9] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256). JMLR Workshop and Conference Proceedings.
[10] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).
[11] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology, 2020*, 4(12), 310-316.
[12] Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination press.