Menemui Matematik (Discovering Mathematics) 46(1) (2024) 47-73

Menemui Matematik
(Discovering Mathematics)

journal homepage: https://myjms.mohe.gov.my/index.php/dismath/

Hyperparameter Tuning of Deep Neural Network in Time Series Forecasting

Kelly Pang Li Xiang[1], Syafrina Abdul Halim[2*] and Nur Haizum Abd Rahman[3]

[1,2]*Department of Mathematics, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor*
[3]*Centre for Mathematical Sciences, Universiti Malaysia Pahang Al-Sultan Abdullah, Lebuh Persiaran Tun Khalil Yaakob, Kuantan 26300, Pahang,*

[1]205851@student.upm.edu.my, [2*]syafrina@upm.edu.my, [3]haizum@umpsa.edu.my
*Corresponding author

Received:01 Jun 2024
Accepted:20 August 2024

ABSTRACT

Deep Artificial Neural Network (DANN) is a type of Artificial Neural Network (ANN) with multiple hidden layers, making them a 'deep' form of ANN. Since ANN is a type of Deep Neural Network (DNN), DANNs fall under the broader DNN category and are widely used in time series forecasting. The performance of DANN is highly dependent on the choice of hyperparameters. Random selection of the hyperparameters may increase DANN's forecasting error. Hence, this study aims to optimize the performance of DANN in time series forecasting by tuning two important hyperparameters: the number of epochs and batch size. In this study, DANN with 1, 10, 20, 50 and 100 epochs, and batch sizes of 32 and 64 are used to grid search and form different combinations of hyperparameters. The performances of each model are evaluated and compared based on the mean square error (MSE) and mean absolute error (MAE). In addition, mean absolute percentage error (MAPE) is used to compare the performance of the DANN model on high-frequency and low-frequency time series data. Our study use simulated and real-life data to reveal the performance of the DANN model. The results show more than one epoch is needed to provide good performance. Specifically, analysis of simulated data consistently suggests that 10 epochs offer optimal results. Similarly, 10 epochs yield optimal results for low-frequency real-life data, while high-frequency real-life data prefers 100 epochs. Additionally, the finding indicates that batch sizes of 32 and 64 are optimal when used in different combinations. Hence, this study suggests that, in starting the learning process, it is crucial to perform hyperparameter tuning. This step ensures the selection of appropriate hyperparameter values, which significantly impact the learning outcome of a DNN model, leading to improved forecast accuracy results.

Keywords: batch size, Deep Artificial Neural Network, epoch, forecasting, hyperparameter

INTRODUCTION

Time series analysis involves exploring, modelling, and predicting sequential observations collected over time. While traditional statistical methods have long been employed for such tasks, they often struggle to capture the intricate dependencies and nonlinear patterns present in time series data. In recent years, the advent of Deep Neural Networks (DNNs) has offered a promising alternative, revolutionising the field with their ability to handle complex temporal relationships. DNNs, characterised by their multi-layered architecture, have emerged as powerful tools for time series analysis due to their capability to automatically learn and extract hierarchical features from data.

Drawing inspiration from the structure and function of the human brain, these networks excel at capturing both short-term fluctuations and long-term trends within sequential data, making them well-suited for tasks such as forecasting and anomaly detection. Researchers have utilised DNN to forecast time series data in areas such as wind power and electric load (Lin et al., 2021; Gasparin et al., 2022). Within the realm of DNNs, various architectures such as Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs) offer distinct advantages for different types of time series data. ANNs serve as the foundational framework for DNNs, while CNNs specialise in extracting spatial and temporal features, and RNNs are adept at capturing sequential dependencies.

Despite their effectiveness, the performance of DNNs heavily depends on the careful tuning of hyperparameters. Optimising hyperparameters through techniques like grid search or Bayesian optimisation is essential to ensure that the DNN achieves its maximum potential. By fine-tuning these parameters, researchers can enhance the model's performance and generalisation capabilities, thereby improving its ability to make accurate predictions on unseen data. Hyperparameters such as the number of epochs, is defined as the number of iterations of the learning algorithm that goes through the entire training dataset, and the batch size, referred to as the number of samples needed to be processed before updating internal model hyperparameters, play a crucial role in determining the model's convergence and predictive accuracy (Ngoc et al., 2021). Epoch plays a crucial role in adjusting the network's parameters for improved performance. Increasing the number of epochs allows the learning algorithm to minimise model error effectively. However, choosing the appropriate number of epochs is a trade-off between computation time and achieving the desired accuracy. Studies have shown that training on larger datasets with more epochs can significantly improve performance (Radford et al., 2018, Devlin et al., 2019). However, a study suggested that one epoch is enough to train on a larger dataset, unlike the current practice. Thus, it comes to an interest in knowing the optimal number of epochs that should be used in a DNN model. The batch size hyperparameter refers to how a small batch size can introduce noise to gradient computation, aiding in escaping sharp local minima and finding more generalised minima for improved performance (Smith et al., 2020). However, larger batch sizes offer steady convergence and acceptable test performance. Additionally, the importance of selecting batch sizes that maximise Graphics Processing Unit (GPU) processing capabilities further emphasises the need for careful consideration and experimentation (Kandel and Castelli, 2020). In conclusion, DNNs offer a promising avenue for time series analysis, providing a flexible and powerful framework for capturing complex temporal patterns. However, achieving optimal performance requires careful consideration and optimisation of hyperparameters, highlighting the importance of rigorous experimentation and tuning in the development of DNN-based models for time series forecasting and analysis. Thus, optimising the hyperparameters, especially the epochs and batch size, is essential for achieving the best performance.

This research aims to determine the optimal combination of epochs and batch size for Deep Artificial Neural Networks (DANN) and evaluate their performance on high-frequency versus low-frequency data. By grid searching these hyperparameters and comparing forecasting accuracy, the study seeks to enhance DANN predictions, leading to more efficient learning and improved forecasting. Furthermore, analysing DANN performance across different data frequencies will provide insights into the model's generalisation capabilities and help stakeholders make informed decisions based on its effectiveness in various time horizons and application scenarios. This research is divided into three parts for clarity and efficiency. First, a simulation study generates high-frequency and low-frequency time series data to tune epochs and batch size using grid search, with performance assessed through mean squared error (MSE), mean absolute error (MAE), and mean absolute percentage error (MAPE). The goal is to identify the optimal hyperparameter values and evaluate whether the DANN performs better on high-frequency or low-frequency data. Second,

the study applies this methodology to real-life data sets, tuning the DANN model and determining optimal hyperparameters based on forecasting performance metrics (MSE, MAE), and comparing results using MAPE. Finally, the study compares the DANN's forecasting accuracy and effectiveness on real-life data with the simulation results, assessing performance across different time horizons to evaluate the model's practical applicability.

MATERIALS AND METHODS

Exploratory Data Analysis

Exploratory data analysis (EDA) is a more systematic and comprehensive approach to analyze data. It aims to uncover patterns and relationships such as trends, seasonality and cyclical patterns in the data, identify outliers or missing values, and gain insights that can guide subsequent analysis or hypothesis generation.

Data Preparation

Both simulation study and real data analysis involving high-frequency data and low-frequency data are conducted in this research. The simulated data are generated using R programming by setting the frequency at 12 and 52 to resemble the time series data with seasonal variations collected monthly (low-frequency data) and weekly (high-frequency data) for each year. For the real-life data, the US airline passenger data taken from an inbuilt dataset of R called AirPassengers are used to represent the low-frequency data, while the 208 weekly mean temperature data derived from the Delhi Climate dataset, which comprises 1461 daily observations spanning from January 1, 2013, to December 31, 2016, sourced from Kaggle, are used as the high-frequency data. Before modelling, the time series data is divided into training and testing sets based on the 80% and 20% ratio, respectively. The training dataset is used during the neural network training stage to update the network weights and biases and determine the gradient to produce a well-generalized network model. The testing dataset is not involved in the training stage but to assess the performance of the model.

Min-Max Normalisation

Data normalization is employed to enhance the efficiency of neural network training. Various pre-processing methods have been developed for this purpose, and one of the most used techniques is min-max normalization. This technique normalizes the dataset into a value range between 0 and 1. The normalizing process is usually done before the training stage, where the processed data is fed into the network. This simplifies the computation and shortens the time for weight updating. Thus, it leads to convergence towards the minimum and improves the effectiveness of the training process. The formula of min-max normalization is as below,

$$x' = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (1)$$

where x' represents the new value in the range between 0 and 1; x_i represents the original data; x_{max} represents the maximum values of the data; and x_{min} represents the minimum values of the data (Aslam et al., 2020).

Walk-Forward Validation

Walk-forward validation is a method in which the model generates a forecast for each observation in the testing dataset, one at a time. Subsequently, the actual observation corresponding to each forecast is incorporated into the testing dataset, thus updating the dataset available for model predictions. The actual observation for a given time step is utilised as part of the input for predicting

the subsequent time step. As the walk-forward validation progresses, the observation list is expanded by appending the actual values corresponding to each prediction made. This allows for a comprehensive comparison between the model's predictions and the true values from the testing set. The rolling of data using the walk-forward validation methodology is as follows (Table 1),

Table 1: The Rolling of Data in the Walk-Forward Validation Methodology

History Data	Predictions
[Months]	Month1
[Months + Month1]	Month2
[Months + Month2]	Month3

Deep Neural Network

a) Deep Neural Network Model Architecture

Deep neural network (DNN) is a highly effective class of machine learning (ML) algorithms that involve the stacking layers of neural network layers in both depth and width, creating compact architectures (Mahmood et al., 2017). Artificial Neural Network (ANN) which is one of the most used DNN models, is described as a flexible computational model which can capture the non-linearity of the data and does not require any prior assumption in modelling (Zhang, 2023). ANN is selected for this research due to its ability to handle and capture non-linear relationships between input and output variables (Zupan, 1994). Additionally, ANN has demonstrated reliable performance in forecasting both short-term and long-term outcomes (Kotur and Zărkovĭc, 2016; Ziari et al., 2016). There are different types of ANN architectures, such as Multilayer Perceptron (MLP) and Feed-Forward Neural Network (FFNN). MLP was focused on this study as it has a structure very similar to DNN, that is consisting of an input layer, one or more hidden layers, and an output layer. The architecture of the MLP network that has only one hidden layer is as in Figure 1,

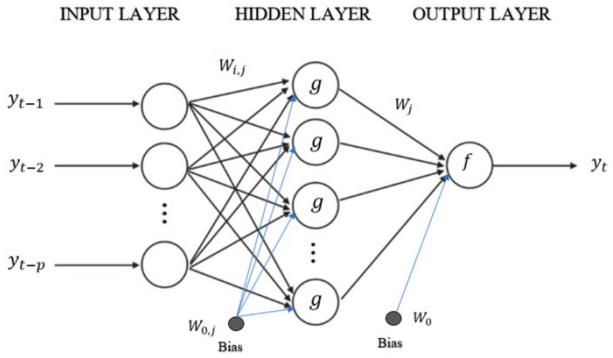


Figure 1: Architecture of Multilayer Perceptron (MLP)

At the input layer phase, the MLP is fitted with the past lagged value of real data, $y_{t-1}, \cdots, y_{t-p}$ as an input vector, resulting in p number of nodes linked to the hidden layer through connection weights, $W_{i,j}$ for $i = 1, 2, \cdots, p$ and $j = 1, 2, \cdots, q$. Each connection weight represents the strength of the connection between an input node and a hidden layer node. There is also a bias unit, $W_{0,j}$ for $j = 1, 2, \cdots, q$ associated with each hidden layer node. The bias unit provides an additional adjustable parameter for adjusting the behaviour of the hidden layer. The activation function within the hidden layer, g performs a calculation involving the weighted sum of input and biases. This introduces the non-linearity to the model. The result of the activation function is then multiplied by the connection weights W_j for $j = 1, 2, \cdots, q$ and summed with the bias W_0. This process is repeated q times, corresponding to each node in the hidden layer. The output layer uses the outputs from the hidden layer to compute the predicted values. Typically, the activation function in the output layer

is linear, aiming to produce the desired output without introducing non-linearity. In summary, the relationship between the input and output layers can be represented in the following formula,

$$y_t = W_0 + \sum_{j=1}^{q} W_j \cdot g\left[W_{0,j} + \sum_{i=1}^{p} W_{i,j} \cdot y_{t-i}\right] + \varepsilon_t \quad (2)$$

where ε_t is the error term, while $W_{i,j}$ and W_j for $i = 1, 2, \cdots, p$ and $j = 1, 2, \cdots, q$ are the connection weights (Khashei and Hajirahimi, 2019).

When it comes to DNN, it must consist of more than one hidden layer. Hence, the architecture of a DNN model will be as in Figure 2,

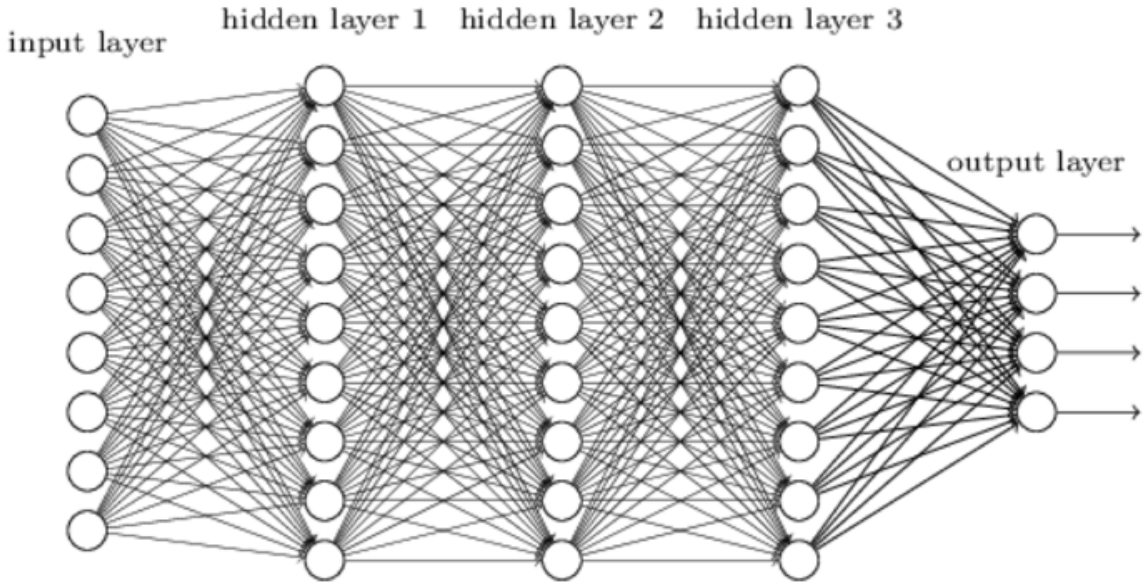


Figure 2: Architecture of Deep Neural Network (DNN)

In a DNN with multiple hidden layers, each hidden layer performs its own set of computations, including the weighted sum of inputs, the application of an activation function, and the propagation of the outputs to the next layer. These computations are performed sequentially across the layers, with each layer taking the outputs of the previous layer as inputs. The formulas and calculations in each hidden layer are similar to those in a single-layer MLP, involving the weighted sum of inputs, biases, and the application of an activation function. However, the outputs of one hidden layer serve as inputs to the next hidden layer, and this process is repeated for each hidden layer until reaching the output layer.

b) Hyperparameter

- **Optimiser Algorithm**

The Adam algorithm, developed by Kingma and Ba, is a hybrid of the AdaGrad and RMSProp algorithms (Kingma and Ba, 2015; Nwankpa, 2020). Unlike traditional Stochastic Gradient Descent (SGD) algorithms that use a single learning rate for all weight updates, the Adam algorithm is an adaptive optimization algorithm. It calculates individual adaptive learning rates for different parameters based on estimated gradients. Adam keeps track of two important quantities, v_t, which represents an exponentially decaying average of past squared gradients, and m_t, which represents an exponentially decaying average of past gradients (Nwankpa, 2020). The formulas for these two quantities are as below,

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{3}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{4}$$

where g_t represents the gradient at time step t. β_1 and β_2 are hyperparameters that control the exponential decay rate of these moving averages, typically in the range of [0, 1]. To prevent any bias towards zero in m_t and v_t, a bias correction mechanism is incorporated,

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{5}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{6}$$

The gradient updates are estimated directly from the running average of this first and second moment of a gradient to produce the update rule as

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon} \widehat{m}_t \tag{7}$$

where the default values are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. The pseudocode for Adam algorithm is as in Figure 3,

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 3: Pseudocode of Adam Algorithm

- **The Number of Epochs**

The determination of the optimal number of epochs in hyperparameter tuning for DNN in time series forecasting involves several key steps. Initially, the range of epoch values to be explored through grid search is defined. It is common to start with a reasonably small number of epochs, such as 10 or 20, to get an initial sense of the model's performance, and then continue with a larger number of epochs. However, a study suggested that one epoch is enough to train on a larger dataset, unlike the current practice. So, 1, 10, 20, 50 and 100 range is set for the number of epochs in this study.

- **The Batch Size**

A systematic approach is implemented in this study to determine the most suitable batch size. Initially, a range of batch sizes is defined for exploration using grid search. While a default batch size of 32 is recommended, it is important to consider alternative options such as 64 and 128. It is recommended to commence with smaller batch sizes, such as 32 or 64, and gradually increase them until desirable outcomes are achieved. Moreover, the number of batch sizes should be a power of two to maximize the Graphics Processing Unit (GPU)'s processing capabilities (Kandel and Castelli, 2020). Hence, this study examines batch sizes of 32 and 64.

c) Backpropagation Training Methodology

The backpropagation algorithm is a fundamental component of neural networks and is widely used, especially for feedforward neural networks. It is used to train a neural network and update the weights and biases by minimising the cost function. The quadratic cost function, C that is used to quantify the discrepancy between the predicted and actual outputs can be written as

$$C = \frac{1}{2n}\sum_{j=1}^{L}\left(y_j - a_j^L\right)^2 \tag{8}$$

where n represents the total number of training samples, L is the number of output nodes in the neural network, y_j represents the actual output for the j^{th} output node and a_j^L represents the activation (predicted output) of the j^{th} neuron in the output layer.

- An equation for the error in the output layer,

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\sigma'\left(z_j^L\right) \tag{9}$$

where σ represents the activation function; and z_j^L represents the weighted input to the j^{th} neuron in the L^{th} level.

- An equation for the error δ^l in terms of the error in the next layer,

$$\delta^l = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l) \tag{10}$$

where $(w^{l+1})^T$ represents the transpose of the weight matrix $w^{l+1}$for the $(l+1)^{th}$ layer; δ^{l+1} represents the error in the $(l+1)^{th}$ layer; σ represents the activation function; $\odot$ represents the Hadamard product; and z^l represents weighted input to the neuron in the l^{th} layer.

- An equation for the rate of change of the cost with respect to any bias in the network,

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{11}$$

where b_j^l represents the bias vector of j^{th} neuron in the l^{th} layer; and δ_j^l represents the error of j^{th} neuron in the l^{th} layer.

- An equation for the rate of change of the cost concerning any weight in the network,

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{12}$$

where w_{jk}^l represents the weight of j^{th} row and k^{th} column of neuron in the l^{th} layer; a_k^{l-1} represents the activation of the k^{th} neuron in the $(l-1)^{th}$ layer; and δ_j^l represents the error of j^{th}

neuron in the l^{th} layer (Nielson, 2015). While traditional optimization techniques like gradient descent are commonly employed, this study deviates from that approach. Instead of gradient descent, the Adam optimizer is utilized due to its computational efficiency and effectiveness in handling complex optimization problems (Kingma and Ba, 2015). The Adam optimizer is elaborated on in the previous section.

Forecasting Accuracy Measures

The testing dataset is used to evaluate forecasting accuracy after the model has been trained on the training dataset. Once the model is trained and able to generate predictions, the testing dataset helps assess how well these predictions match actual values that the model has not seen before. To evaluate the accuracy of the forecasts obtained from the fitted models, three commonly used metrics are computed: Mean Squared Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). These measures provide insights into the model's performance and its ability to make accurate predictions. MSE is a measure of the variability of errors in the forecasted values. It quantifies the average squared difference between the predicted values and the actual values. A smaller MSE indicates better prediction performance. The formula for MSE is as below,

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2, \tag{13}$$

where $\hat{y}_t$ represents the predicted values.

However, the MSE penalizes extreme errors and is sensitive to data transformations and changes in scale (Golingay, 2020). In contrast, MAE does not penalize extreme errors. It measures the average absolute deviation of the forecasted values from the original values. Similar to MSE, the smallest MAE indicates better prediction performance. The formula for MAE is as below,

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t| \tag{14}$$

MAPE is a measure of the accuracy of a forecasting method in predicting values. It is preferred due to its advantages of scale-independency and interpretability (Kim and Kim, 2016). MAPE indicates how much error in predicting compared with the real value. The formula of MAPE is as below,

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{y_t} \text{x } 100\% \text{ where } y_t > 0 \tag{15}$$

By calculating and analysing these accuracy measures, researchers can assess the performance of models in forecasting time series data and compare the effectiveness of different models or parameter configurations.

RESULTS AND DISCUSSION

Simulation Study

Both low-frequency and high-frequency time series data are simulated using three SARIMA models with different coefficients and sample sizes with replication of 500 times using R programming. The frequency is set at 12 and 52, so the simulated data resembles the time series data with seasonal variations collected monthly and weekly for each year. Hence, the sample sizes need to be a multiple of 12 and 52. Since all data are simulated by specifying their coefficients and models in R programming, white noise is assumed to exist in the data, and the data is assumed to be stationary after detrended and deseason, so that the stationary condition is fulfilled (Gikungu et at., 2015). After detrended and deseason, the data is split into an 80:20 ratio before being trained and tested using the DANN model.

a) Deep Artificial Neural Network Modelling

A deep neural network is an extension of a simple neural network which consists of one input layer, one output layer, and at least two hidden layers. Two hidden layers of neural networks are used to improve performance (Thomas et al., 2016; Thomas et al., 2017; Guliyev and Ismailov, 2018). So, two hidden layers are used to investigate the simulated data. The simulated data are considered as univariate time series data, so one input node and one output node are used in the DANN model. The number of nodes selected is 100 and the activation function used is ReLU with the hope that the neuron network fulfilled the Universal Approximation Theorem, which states that a neural network with at least one hidden layer of a sufficient number of neurons with non-linear activation functions can approximate any continuous function to an arbitrary level of accuracy (Cybenko, 1989; Hornik et al., 1989; Sonoda and Murata, 2017). For the hyperparameters of interest, the number of epochs to be investigated is 1, 10, 20, 50 and 100 in this study as there was a suggestion that one epoch is enough to train on a large dataset, but a common way is to start with a reasonably small number of epochs, such as 10 or 20, and then continue with a larger number of epochs. Batch sizes 32 and 64 are used in this study as the batch size should be at least one but not greater than the number of samples in the training set (Table 2). Moreover, the MSE loss function and Adam optimisation are also applied in this study. The grid search values for DANN model for simulated data is as Table 2.

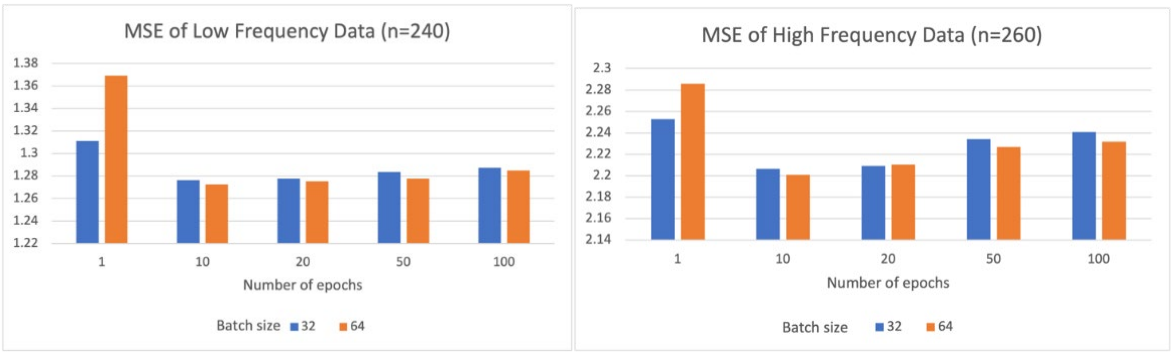
Table 2: Grid Search Values for DANN model for Simulated Data

No	Hyperparameters	Values
1	Input Node	1
2	Hidden Layers	2
3	Hidden Nodes	100
4	Output Node	1
5	Activation Functions	ReLU
6	Epochs	1, 10, 20, 50, 100
7	Batch Size	32, 64

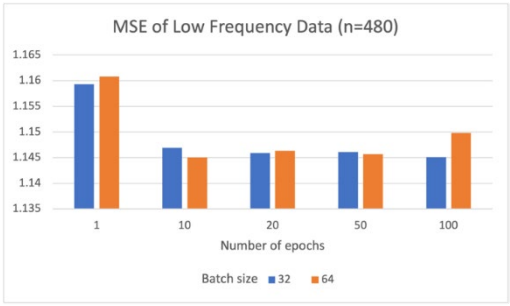
Before running the datasets through the DANN model, data transformation is carried out to enhance the learning process of the models. Min-max normalisation is conducted to ensure the data ranges from 0 to 1. The average MSE and MAE are then computed for the 500 replicates for each sample size. This process ensures the consistency of the results besides dealing with the stochastic nature of the neural network algorithm.

b) Best Hyperparameters of Interest

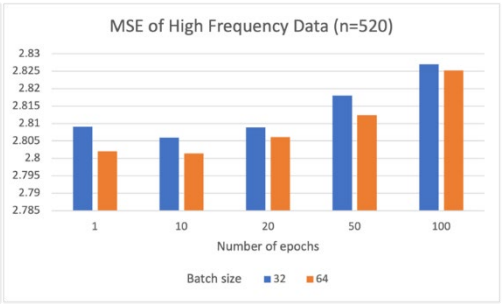
The results below show the bar plot of MSE and MAE values for both high-frequency and low-frequency data sorted in ascending sample size form (Figure 4),



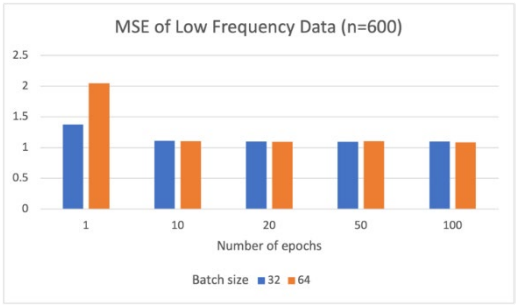
(a) (b)


MSE of Low Frequency Data (n=480)
1.165
1.16
1.155
1.15
1.145
1.14
1.135
1
10
20
50
100
Number of epochs
Batch size 32 64

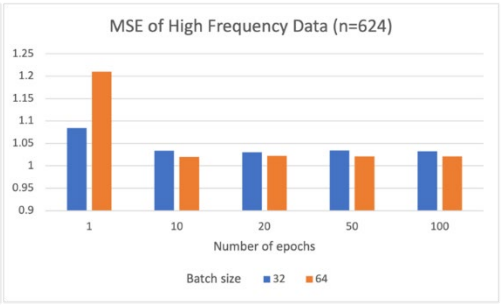
(c)


MSE of High Frequency Data (n=520)
2.83
2.825
2.82
2.815
2.81
2.805
2.8
2.795
2.79
2.785
1
10
20
50
100
Number of epochs
Batch size 32 64

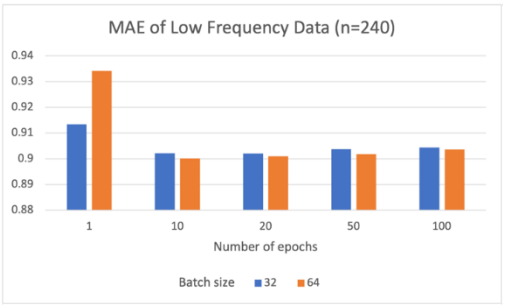
(d)


MSE of Low Frequency Data (n=600)
2.5
2
1.5
1
0.5
0
1
10
20
50
100
Number of epochs
Batch size 32 64

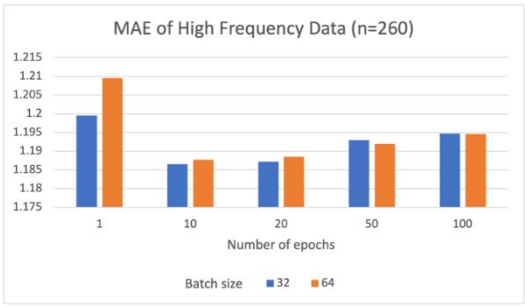
(e)


MSE of High Frequency Data (n=624)
1.25
1.2
1.15
1.1
1.05
1
0.95
0.9
1
10
20
50
100
Number of epochs
Batch size 32 64

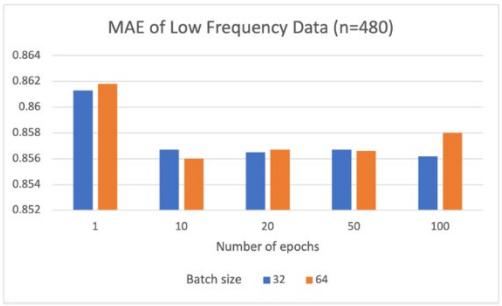
(f)


MAE of Low Frequency Data (n=240)
0.94
0.93
0.92
0.91
0.9
0.89
0.88
1
10
20
50
100
Number of epochs
Batch size 32 64

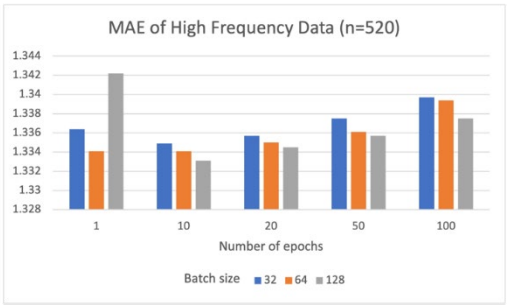
(g)


MAE of High Frequency Data (n=260)
1.215
1.21
1.205
1.2
1.195
1.19
1.185
1.18
1.175
1
10
20
50
100
Number of epochs
Batch size 32 64

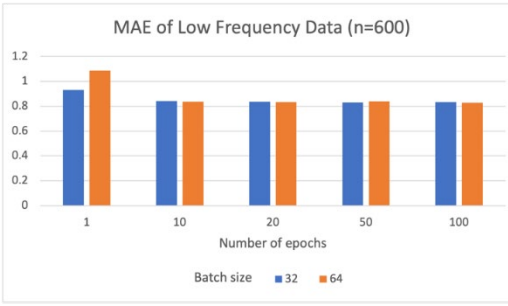
(h)


MAE of Low Frequency Data (n=480)
0.864
0.862
0.86
0.858
0.856
0.854
0.852
1
10
20
50
100
Number of epochs
Batch size 32 64

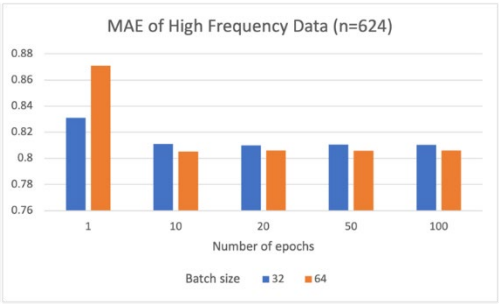
(i)


MAE of High Frequency Data (n=520)
1.344
1.342
1.34
1.338
1.336
1.334
1.332
1.33
1.328
1
10
20
50
100
Number of epochs
Batch size 32 64 128

(j)


MAE of Low Frequency Data (n=600)
1.2
1
0.8
0.6
0.4
0.2
0
1
10
20
50
100
Number of epochs
Batch size 32 64

(k)


MAE of High Frequency Data (n=624)
0.88
0.86
0.84
0.82
0.8
0.78
0.76
1
10
20
50
100
Number of epochs
Batch size 32 64

(l)

Figure 4: MSE and MAE for Both Low-Frequency and High-Frequency Simulated Data,

(a) MSE for SARIMA$(0,0,1)(0,1,1)_{12}$ with sample size $n = 240$,

(b) MSE for SARIMA$(1,1,0)(0,1,0)_{52}$ with sample size $n = 260$,

(c) MSE for SARIMA$(0,1,1)(0,1,1)_{12}$ with sample size $n = 480$,

(d) MSE for SARIMA$(0,0,1)(1,1,0)_{52}$ with sample size $n = 520$,

(e) MSE for SARIMA$(1,1,0)(1,1,0)_{12}$ with sample size $n = 600$,

(f) MSE for SARIMA$(1,0,0)(0,1,0)_{52}$ with sample size $n = 624$,

(g) MAE for SARIMA$(0,0,1)(0,1,1)_{12}$ with sample size $n = 240$,

(h) MAE for SARIMA$(1,1,0)(0,1,0)_{52}$ with sample size $n = 260$,

(i) MAE for SARIMA$(0,1,1)(0,1,1)_{12}$ with sample size $n = 480$,

(j) MAE for SARIMA$(0,0,1)(1,1,0)_{52}$ with sample size $n = 520$,

(k) MAE for SARIMA$(1,1,0)(1,1,0)_{12}$ with sample size $n = 600$,

(l) MAE for SARIMA$(1,0,0)(0,1,0)_{52}$ with sample size $n = 624$

Table 3: MSE Low-Frequency and High-Frequency Simulated Data

Sample Size	Batch Size	Epoch				
		1	10	20	50	100
240	32	1.3111	1.2764	1.2779	1.2835	1.2873
	64	1.3693	1.2725	1.2752	1.2779	1.2847
260	32	2.2526	2.2065	2.2089	2.234	2.2406
	64	2.2858	2.2007	2.2105	2.2266	2.2318
480	32	1.1593	1.1469	1.1459	1.1461	1.1451
	64	1.1608	1.145	1.1463	1.1457	1.1498
520	32	2.8091	2.8059	2.8089	2.818	2.827
	64	2.802	2.8014	2.8061	2.8124	2.8252
600	32	1.3768	1.1112	1.1	1.0949	1.0999
	64	2.0474	1.108	1.0946	1.1071	1.0864
624	32	1.0848	1.0339	1.0306	1.034	1.0325
	64	1.21	1.02	1.0222	1.0212	1.0214

Table 4: MAE Low-Frequency and High-Frequency Simulated Data

Sample Size	Batch Size	Epoch				
		1	10	20	50	100
240	32	0.9134	0.9021	0.902	0.9038	0.9044
	64	0.9341	0.9001	0.9009	0.9018	0.9037
260	32	1.1995	1.1865	1.1872	1.1929	1.1947
	64	1.2095	1.1877	1.1885	1.1919	1.1946
480	32	0.8613	0.8567	0.8565	0.8567	0.8562
	64	0.8618	0.856	0.8567	0.8566	0.858
520	32	1.3364	1.3349	1.3357	1.3375	1.3397

	64	1.33415*	1.33413*	1.335	1.3361	1.3394
600	32	0.9308	0.8399	0.8349	0.8316	0.834
	64	1.0868	0.8373	0.8328	0.8378	0.8292
624	32	0.8311	0.8111	0.8098	0.8107	0.8104
	64	0.8709	0.8052	0.8061	0.8058	0.806

Tables 3 and 4 above show the summary of MSE and MAE values for both low-frequency and high-frequency simulated data. From the results shown, almost all MSE and MAE values for those using one epoch (except for sample size n = 520) have the largest values compared to the MSE and MAE values of other epochs. Although the MSE and MAE values for those using one epoch in sample size n = 520 are of smaller values, they are not the lowest MSE and MAE values. The values marked with asterisk are of 5 decimal places so that their actual values are pictured, as they are too close to each other. Thus, by comparing the overall performance of one epoch and other epochs, it is suggested that one epoch is insufficient to have a good performance to forecast the time series data with seasonal variations. The results of the MSE and MAE values across the epochs from 1 epoch to 100 epochs differ as the simulated models and coefficients used are different. The MSE values are decreasing from 1 epoch to 10 epochs then increasing from 10 epochs to 100 epochs (batch sizes of 32 and 64 for n = 240, batch sizes of 32 and 64 for n = 260, batch size of 32 and 64 for n = 520), while the MSE values are decreasing from 1 epoch to 20 epochs then increasing from 20 epochs to 50 epochs and finally decreasing from 50 epochs to 100 epochs (batch size of 32 for n = 480, batch size of 64 for n = 600, batch size of 32 for n = 624).

The MSE values are decreasing from 1 epoch to 10 epochs, then increasing from 10 to 20 epochs, followed by a decreasing trend from 20 epochs to 50 epochs and finally increasing from 50 epochs to 100 epochs (batch size of 64 for n = 480, batch size 64 for n = 624). For sample size n = 600, the MSE values for those using batch size of 32 are decreasing from 1 epoch to 50 epochs then increasing from 50 epochs to 100 epochs. The MAE values are decreasing from 1 epoch to 10 epochs then increasing from 10 epochs to 100 epochs (batch size of 64 for n = 240 and batch sizes of 32 and 64 for n = 260, batch sizes of 32 and 64 for n = 520), while the MAE values are decreasing from 1 epoch to 20 epochs then increasing from 20 epochs to 100 epochs (batch size of 32 in n = 240). The MAE values are decreasing from 1 epoch to 20 epochs then increasing from 20 epochs to 50 epochs, and finally decreasing from 50 epochs to 100 epochs (batch size of 32 in n = 480, batch size of 64 in n = 600, batch size of 32 in n = 624), while the MAE values are decreasing from 1 epoch to 10 epochs, then increasing from 10 to 20 epochs and decreasing from 20 epochs to 50 epochs, at last increasing from 50 epochs to 100 epochs (batch size of 64 in n = 480, batch size of 64 in n = 624). For sample size n = 600, the MAE values for those using batch size of 32 are decreasing from 1 epoch to 50 epochs then increasing from 50 epochs to 100 epochs. Thus, it concludes that the trend of several epochs is unpredictable for time series data with seasonal variation.

In terms of MSE values, for all the batch sizes in the six-sample size, 8 out of 12 suggested that 10 epochs are the optimal number of epochs that gave the lowest MSE values, while 20 epochs and 50 epochs each have only 1 out of 12 prove that they are the optimal number of epochs. 2 out of 12 suggested that 100 epochs are the optimal number of epochs that gave lowest MSE values. In terms of MAE values, for all the batch sizes in the six-sample size, 7 out of 12 suggested that 10 epochs are the optimal number of epochs that gave the lowest MAE values. There are 2 out of 12 MAE values suggesting that 20 epochs are the optimal number of epochs, same does 100 epochs, while 1 out of 12 suggested that 50 epochs are the optimal number of epochs. Thus, in general, the analysis of simulated data consistently suggests that 10 epochs offer optimal results. However, the best number of epochs are inconsistent as the results have also suggested other epochs as the optimal

number of epochs besides 10 epochs. For the batch size, the results show that it is inconsistent since 32 and 64 both can be the best number of batch sizes in different configurations by assessing the MSE and MAE values. When the batch size and number of epochs are constant, the MSE and MAE values no longer show a decreasing trend when the sample size increases. In fact, they fluctuate up and down probably due to different models and coefficients used. In conclusion, the optimal values of batch size and the number of epochs for time series data with seasonal variations are inconsistent by determining through MSE and MAE values, as it depends on the patterns of the time series data. Different models of SARIMA and different coefficients implemented will certainly make the time series data vary in seasonal variations and complexity, making the optimal values of batch size and the number of epochs keep changing to get a better result.

c) Comparison between Low-Frequency and High-Frequency Data

The results below are the MAPE values for low-frequency and high-frequency data (Figure 5),

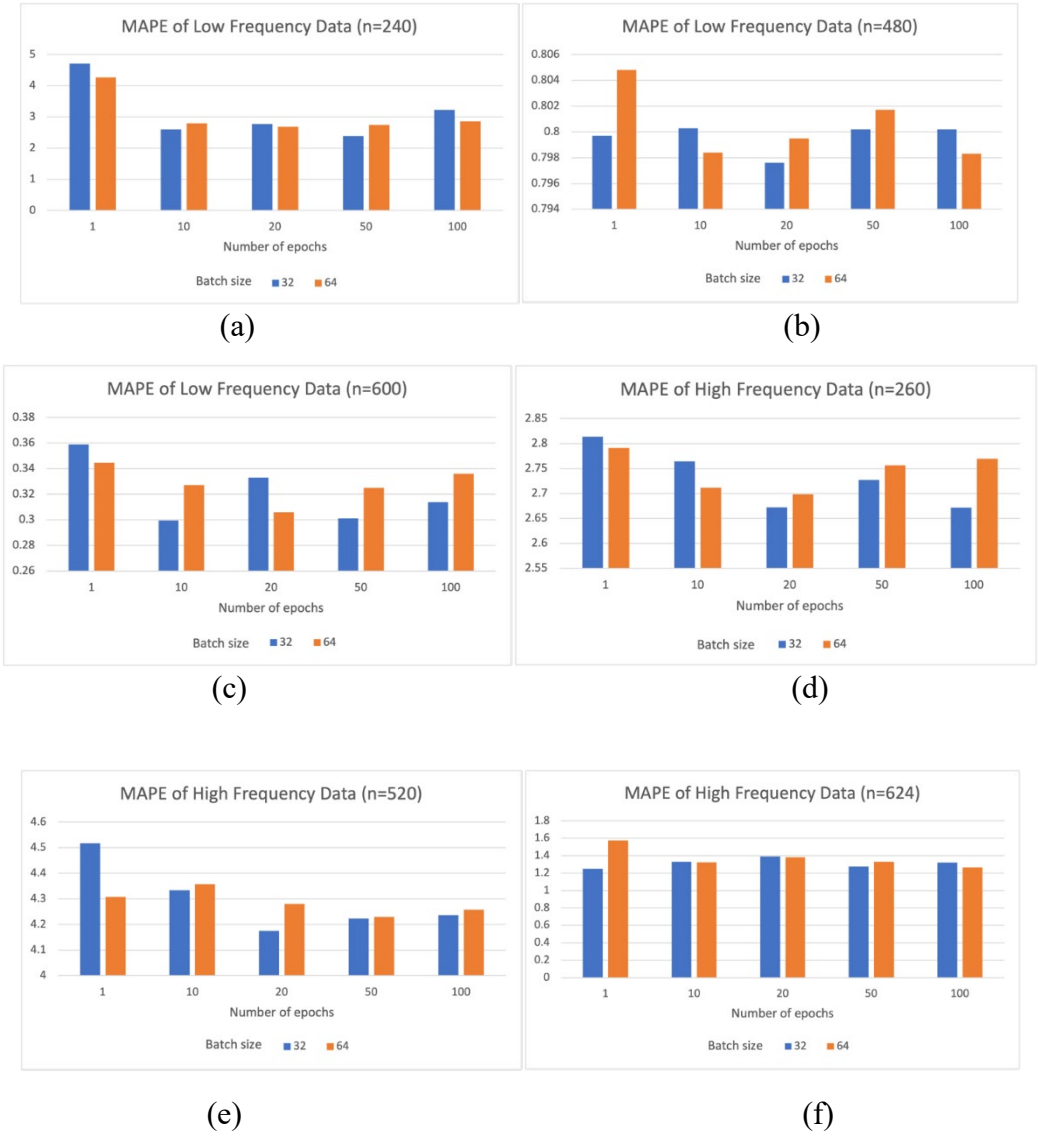


Figure 5: MAPE for Low-Frequency and High-Frequency Simulated Data,
(a) MAPE for SARIMA$(0,0,1)(0,1,1)_{12}$ with sample size $n = 240$,
(b) MAPE for SARIMA$(0,1,1)(0,1,1)_{12}$ with sample size $n = 480$,
(c) MAPE for SARIMA$(1,1,0)(1,1,0)_{12}$ with sample size $n = 600$,

(d) MAPE for SARIMA$(1,1,0)(0,1,0)_{52}$ with sample size $n = 260$,
(e) MAPE for SARIMA$(0,0,1)(1,1,0)_{52}$ with sample size $n = 520$,
(f) MAPE for SARIMA$(1,0,0)(0,1,0)_{52}$ with sample size $n = 624$

The MAPE values for both low-frequency and high-frequency data are grouped into the Tables 5 and 6 below to ease the interpretation. The sample size n = 240 has higher MAPE values than other low-frequency data, so as the sample size n = 520 for high frequency data. However, in overall, DANN performs better on low-frequency data, as its MAPE values, if treated as a whole, are lower than high-frequency data.

Table 5: MAPE for Low-Frequency Simulated Data

Sample Size	Batch Size	Epoch				
		1	**10**	**20**	**50**	**100**
240	32	4.7124	2.6009	2.7789	2.3875	3.2296
	64	4.2718	2.7963	2.6907	2.7413	2.8581
480	32	0.7997	0.8003	0.7976	0.8002	0.8002
	64	0.8048	0.7984	0.7995	0.8017	0.7983
600	32	0.359	0.2994	0.333	0.3011	0.3137
	64	0.3447	0.3272	0.306	0.3251	0.3361

Table 6: MAPE for High-Frequency Simulated Data

Sample Size	Batch Size	Epoch				
		1	**10**	**20**	**50**	**100**
260	32	2.814	2.7646	2.6717	2.7271	2.6711
	64	2.7911	2.711	2.6979	2.7561	2.7695
520	32	4.5171	4.3331	4.1755	4.2238	4.2363
	64	4.3073	4.3568	4.2804	4.2289	4.2584
624	32	1.2476	1.3305	1.3915	1.2753	1.3208
	64	1.5752	1.323	1.3802	1.3295	1.2648

Real Data Analysis

a) Exploratory Data Analysis

- **Low-Frequency Real-life Data**

The US airline passenger data represents low-frequency real-life data in this study. From the EDA results, there are no missing values in the US airline passenger data from the years 1949 to 1961, so as outliers. The box plot of US airline passenger data is as below (Figure 6),

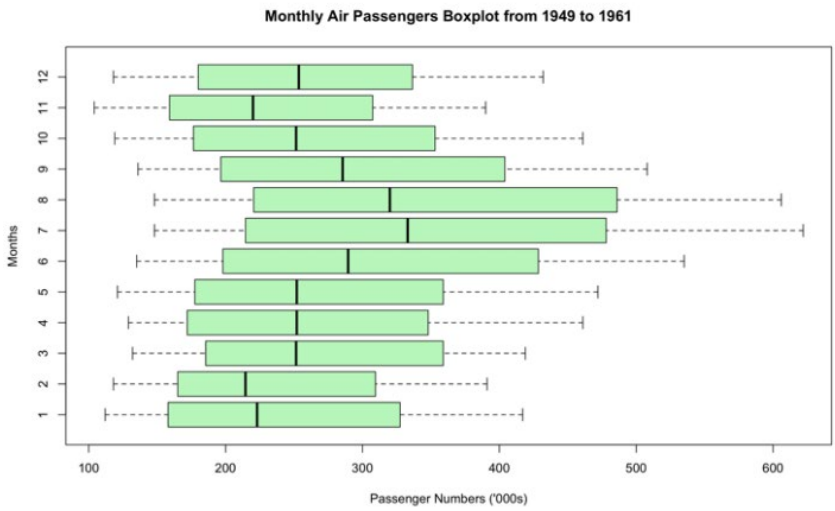


Figure 6: Box Plot of the US Airline Passenger Data

Table 7: Summary Statistics of the US Airline Passenger Data

Minimum	104.0
First Quartile	180.0
Median	265.5
Mean	280.3
Third Quartile	360.5
Maximum	622.0

From the box plot, the US airline passenger data from 1949 to 1961, categorized by month, exhibit right-skewness. This is supported by the summary statistics in Table 7, where the mean is higher than the median, indicating that the distribution has a longer tail on the right side, with a concentration of values towards the lower end and fewer higher values pulling the mean upwards. The summary of the US airline passenger data also shows that the minimum number of airline passengers is 104 thousand while the maximum number of airline passengers is 622 thousand. The decomposition plot of the US airline passenger data is as below (Figure 7),

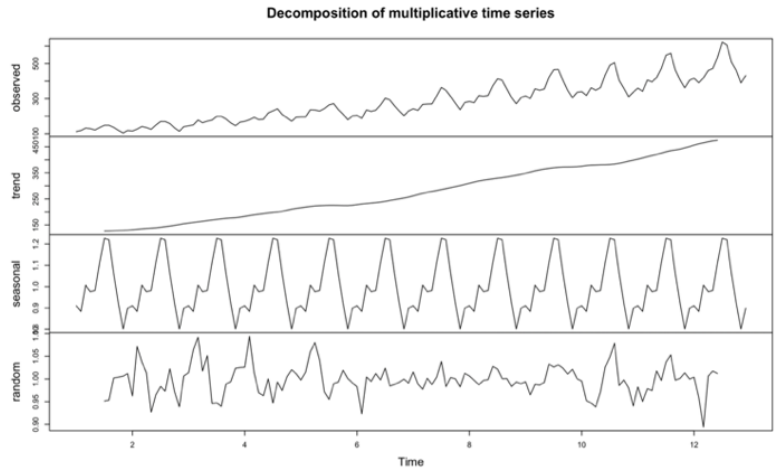


Figure 7: Decomposition Plot of the US Airline Passenger Data

From the decomposition results, the US airline passenger data has an obvious upward trend and seasonal pattern. Also, from the ACF plot, the ACF dies down slowly over a multiple of 12 lags suggesting a seasonality of 12, which indicates that the data exhibits monthly seasonality. The ACF and PACF plots are as below (Figure 8),

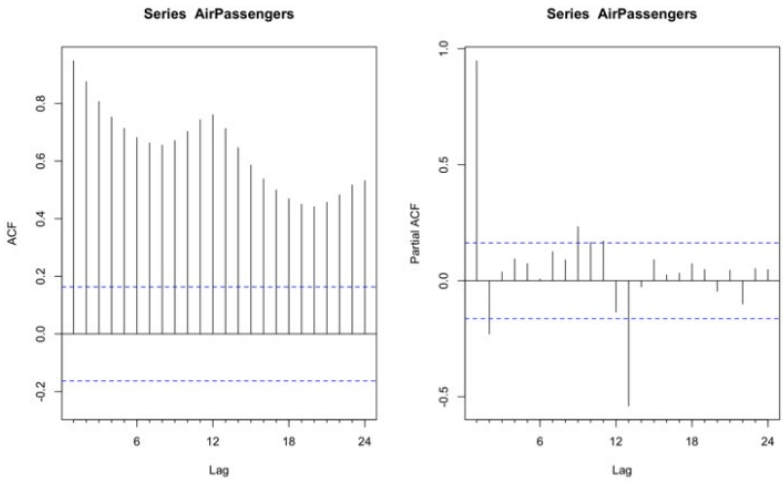


Figure 8: ACF and PACF Plots of the US Airline Passenger Data

The data is not stationary in terms of variance as there is a noticeable variation in the range of values at different points in time. However, a deep neural network (DNN) should be able to deal with the non-seasonality of the data (Musbah et al., 2023). Thus, Box-Cox transformation is not required in this study.

- **High-Frequency Real-life Data**

The Delhi temperature data represents high-frequency real-life data in this study. From the EDA results, there are no missing values in the Delhi temperature data from the years 2013 to 2016, so as outliers. The box plot of Delhi temperature data is as below (Figure 9),

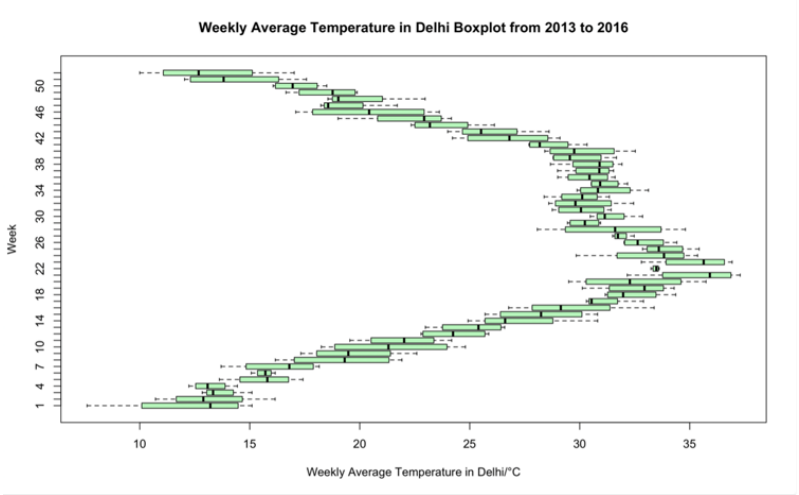


Figure 9: Box Plot of the Delhi Temperature Data

Table 8: Summary Statistics of the Delhi Temperature Data

Minimum	7.605
First Quartile	18.730
Median	28.396
Mean	25.540
Third Quartile	31.209
Maximum	37.293

From the box plot, the Delhi temperature data from 2013 to 2016, when categorized by week, shows variability in skewness, with some periods exhibiting right skewness and others showing left skewness. However, the summary statistics in Table 8 suggest that the data is generally left-skewed, as the median is higher than the mean. The summary of the Delhi temperature data shows that the minimum average weekly temperature is 7.605 °C while the maximum average weekly temperature is 37.293 °C. The time series plot of the Delhi temperature data is as below (Figure 10),

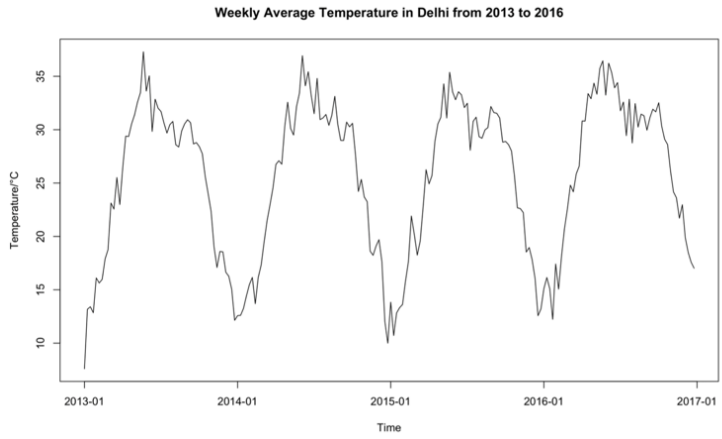


Figure 10: Time Series Plot of the Delhi Temperature Data

From the time series plot of the Delhi temperature data, the seasonal patterns that repeat on a yearly basis can be observed. The data is stationary in terms of variance since the spread of values in the time series remains relatively constant over time. Additive seasonal components are identified since the amplitude of the seasonal pattern remains constant over time and is consistently added to the overall trend of the time series. The ACF and PACF plots of Delhi temperature data is as below (Figure 11),

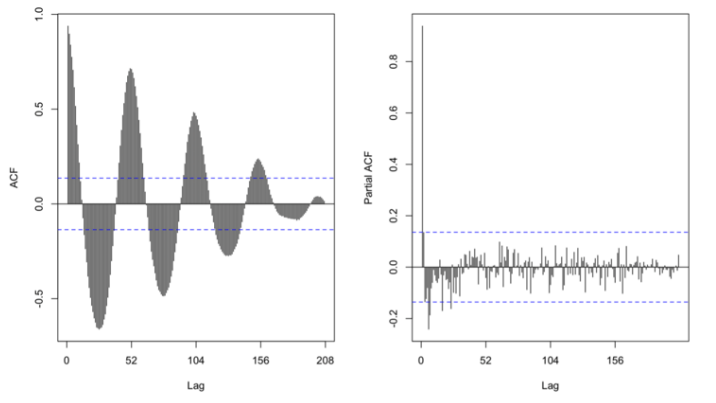


Figure 11: ACF and PACF Plots of the Delhi Temperature Data

From the ACF plot, the ACF dies down slowly over a multiple of 52 lags, suggesting a seasonality of 52, which indicates that the data exhibits weekly seasonality.

b) Data Preparation

- **Data Detrending and Deseasonalisation**

Neural networks may fail to capture the seasonal or trend variations within the unpreprocessed raw time series data. Thus, time series data need to be deseason and detrended, if required, before being trained and tested in a neural network, so that the forecast results can be more accurate (Nelson et al., 1999; Zhang and Qi, 2005). The US airline passenger data possess an upward trend and seasonal variation. Hence, it is detrended using the moving average detrending method and deseason using the differencing method. The time series plot after detrending is as below (Figure 12),

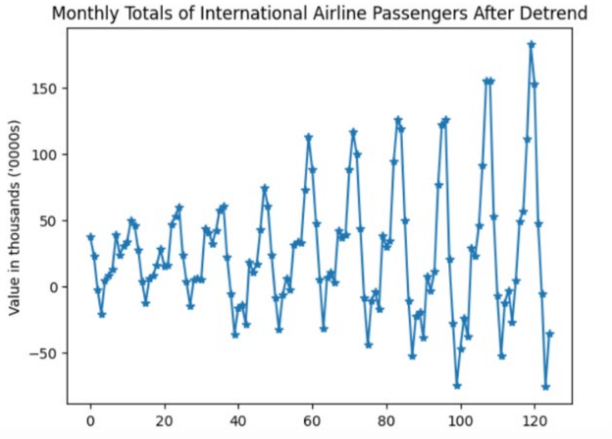


Figure 12: Time Series Plot of the US Airline Passenger Data After Detrending

After detrending using the moving average method, some of the data are removed due to the window size used in the moving average calculation. A commonly used rule of thumb for choosing window size is to use 10% to 25% from the time series data. Thus, in this study, the window size is fixed to 20 to remove the trend effects in the time series data (Azzeh, 2017). Then, seasonal differencing is performed under the grid search method. For the Delhi temperature data, only seasonal differencing under the grid search method is performed since the data do not possess an obvious trend.

- **Data Splitting**

Data splitting is the action of partitioning the data into two parts for cross-validatory (Picard and Berk, 1990). In this research, each data sets were split into training and testing sets in 80:20 ratio after detrending and deseason to evaluate the performance of deep artificial neural network (DANN) model on both high-frequency and low-frequency real-life data. Training data set is applied for DANN model development and training, while the testing data set is used to assess the established model. The real-life data are split into the following Table 9,

Table 9: Sample Compositions in Two Real-life Data Sets

Real-life Data	Original Sample Size	Sample Size After Detrend (if required) and Deseason	Training Sets	Testing Sets
The US Airline Passenger	144	125	100	25
The Delhi Temperature	208	208	167	41

c) Deep Artificial Neural Network Modelling

DNN is an extension of a simple neural network which consists of one input layer, one output layer, and at least two hidden layers. Two hidden layers of neural networks are used so that the performance is better (Thomas et al., 2016; Thomas et al., 2017; Guliyev and Ismailov, 2018). So, two hidden layers are used to investigate the real-life data. The US airline passenger data and the Delhi temperature data are considered univariate time series data, so one input node and one output node are used in the DANN model. The number of nodes selected is 100 and the activation function used is ReLU with the hope that the neuron network fulfilled the Universal Approximation Theorem, which states that a neural network with at least one hidden layer of a sufficient number of neurons with non-linear activation functions can approximate any continuous function to an arbitrary level of accuracy (Cybenko, 1989; Hornik et al., 1989; Sonoda and Murata, 2017). For the hyperparameters of interest, the number of epochs to be investigated is 1, 10, 20, 50 and 100 in this study as there was suggestion that one epoch is enough to train on a large dataset, but a common way is to start with a reasonably small number of epochs, such as 10 or 20, and then continue with a larger number of epochs. Batch sizes of 32 and 64 are used in this study as the batch size should be at least one but not greater than the number of samples in the training set (Table 10). Moreover, MSE loss function and Adam optimisation are also applied in this study.

Table 10: Grid Search Values for DANN model for Real-life Data

No	Hyperparameters	Values
1	Input Node	1
2	Hidden Layers	2
3	Hidden Nodes	100
4	Output Node	1
5	Activation Functions	ReLU
6	Epochs	1, 10, 20, 50, 100
7	Batch Size	32, 64

Before running the datasets through the DANN model, data transformation is carried out to enhance the learning process of the models. Min-max normalisation is conducted to ensure that the data ranges from 0 to 1. Other than that, the DANN for each combination of hyperparameters is run 10 times to ensure consistency as used in Kouassi and Moodley (2020). The average MSE and MAE are then computed. This process is carried out to deal with the stochastic nature of the neural network algorithm. While 10 runs offer a good balance between computational efficiency and reliability, increasing the number of runs could provide a more accurate assessment of performance. However, due to computational and memory constraints, the number of repetitions is limited to 10 to manage resources effectively.

d) Low-Frequency and High-Frequency Real-life Data

Table 11: MSE and MAE for the US Airline Passenger Data

Batch Size	Epoch	MSE	MAE
32	1	364.4928	16.6212
	10	**282.7164**	13.2247
	20	285.4635	13.3866
	50	286.6904	13.4292
	100	290.1145	13.522

64	1	423.5157	18.0942
	10	288.3093	**13.1795**
	20	285.9265	13.275
	50	286.2365	13.3997
	100	295.4707	13.6162

Based on the US airline passenger results as in Table 11, for batch size of 32, the MSE and MAE values are decreasing from 1 epoch to 10 epochs, then increasing from 10 epochs to 100 epochs, while for batch size 64, the MSE values are decreasing from 1 epoch to 20 epochs, then increasing from 20 epochs to 100 epochs. The MAE values for batch size of 64 decrease from 1 epoch to 10 epochs, then increase from 10 epochs to 100 epochs. The MSE values increase with the increase of batch size for all epochs except 50 epochs which decreases as the batch size increases. 1 and 100 epochs have their MAE values increase when the batch size increases, while others have their MAE values decrease when the batch size increases. Overall, for the US airline passenger data, the DANN model with a batch size of 32 and 10 epochs has the lowest MSE values, while the DANN model with a batch size of 64 and 10 epochs have the lowest MAE value. The forecast comparison results are shown in Figure 13 and Figure 14,

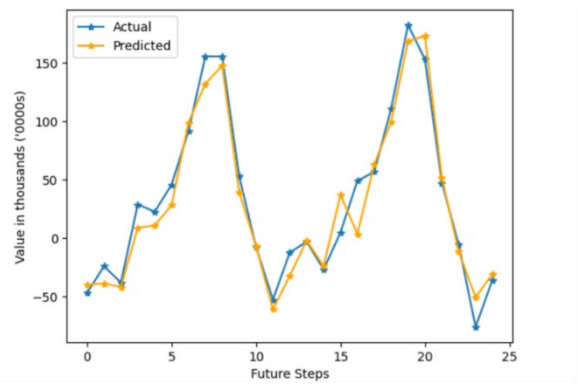


Figure 13. Forecasting Comparison for the Lowest MSE Configuration of the US Airline Passenger Data

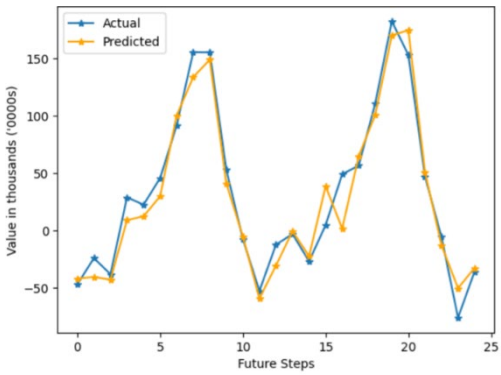


Figure 14. Forecasting Comparison for the Lowest MAE Configuration of the US Airline Passenger Data

Table 12. MSE and MAE for the Delhi Temperature Data

Batch Size	Epoch	MSE	MAE
32	1	5.4951	1.9366
	10	5.2891	1.8621
	20	5.1819	1.8409
	50	5.1897	1.8495
	100	5.1667	1.8506
64	1	6.2976	2.0877
	10	5.1842	1.8438
	20	5.1650	1.8380
	50	5.1853	1.8440
	100	**5.1328**	**1.8374**

Based on the grid search results for the Delhi temperature data as in Table 12, the MSE values for batch size of 32 are decreasing from 1 epoch to 20 epochs, then increasing from 20 epochs to 50 epochs and finally decreasing from 50 epochs to 100 epochs. The MAE values for batch size of 32 are decreasing from 1 epoch to 20 epochs, then gradually increasing from 20 epochs to 100 epochs. For batch size 64, the MSE and MAE values are decreasing from 1 epoch to 20 epochs, then increasing from 20 epochs to 50 epochs, and finally decreasing from 50 epochs to 100 epochs. The MSE and MAE values decrease with the increase of batch size for all epochs except 1 epoch which increases as the batch size increases. Overall, for the Delhi temperature data, the DANN model with batch size of 64 and 100 epochs has the lowest MSE and MAE values. The forecast comparison results are as follows (Figure 15),

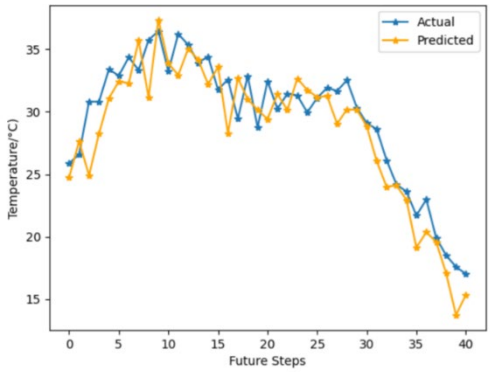


Figure 15. Forecasting Comparison for the Lowest MSE and MAE Configuration of the Delhi Temperature Data

The best MSE and MAE model configurations for real-life low-frequency and high-frequency data are different. Thus, the best combination of the batch size and the number of epochs cannot be determined. It is suggested that large batch sizes offer steady convergence and acceptable test performance while small batch sizes perform better in terms of stability and generalisation (Lin, 2022). Moreover, it is known that increasing the number of epochs allows the learning algorithm to minimise model error effectively, while there was study stated that minimising the number of epochs can achieve the aim of obtaining a minimal root mean square error (RMSE), which results in minimal mean square error (MSE) (Hastomo et al., 2021).

e) Comparison between Low-Frequency and High-Frequency Real-life Data

Table 13. MAPE for Both Low-Frequency and High-Frequency Real-life Data

Batch Size	Epoch	MAPE	
		Low-Frequency Data	High-Frequency Data
32	1	0.8705	0.0687
	10	**0.6025**	0.0652
	20	0.6561	0.0643
	50	0.6826	0.0647
	100	0.6982	0.0648
64	1	0.9531	0.0749
	10	0.6548	0.0645
	20	0.6422	0.0642
	50	0.6753	0.0645
	100	0.6942	**0.0643**

From the results above (Table 13), high-frequency real-life data have lower MAPE values compared to low-frequency real-life data no matter what batch size and epoch are being used. The MAPE values for 1 epoch and 10 epochs for the US airline passenger data increase as the batch size increases, while others decrease as the batch size increases. For the Delhi temperature data, MAPE values decrease with the increase of batch size for all epochs except 1 epoch which increases as the batch size increases. However, the configurations with the lowest MAPE value within them are different. For low-frequency real-life data (the US airline passenger data), the lowest MAPE value is 0.6025, produced by batch size of 32 and 10 epochs, while for high-frequency real-life data (the Delhi temperature data), the lowest MAPE value is 0.0643, produced by batch size of 64 and 100 epochs. The forecasting results are shown in Figure 16 and Figure 17,

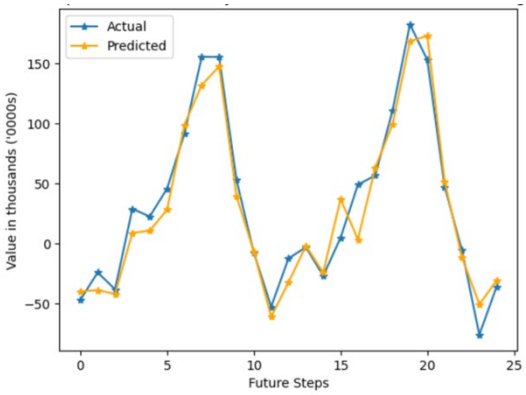


Figure 16. Forecasting Comparison for the Lowest MAPE Configuration of the US Airline Passenger Data

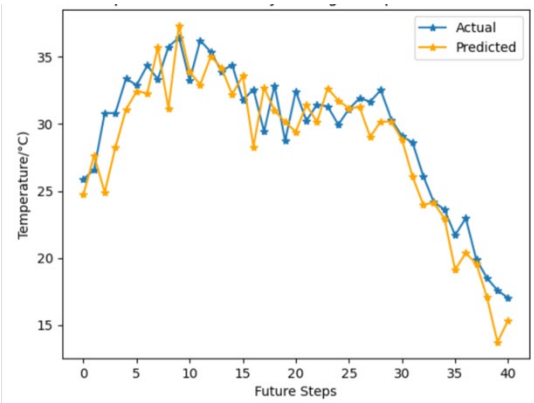


Figure 17. Forecasting Comparison for the Lowest MAPE Configuration of the Delhi Temperature Data

CONCLUSION

This study presents experimental studies on the deep artificial neural network (DANN) application for low-frequency and high-frequency time series data with seasonal variations. Both data are detrended and deseason to avoid instability of the DANN model in directly forecasting seasonal data as proposed by Faraway and Chatfield (1998). The results show that the sample size has limited effects on the DANN's performance. Grid search is used in tuning the DANN model's hyperparameters to optimise the neural network's performance. The main hyperparameter tuned in the study is the number of epochs, where the number of epochs used are 1, 10, 20, 50 and 100. From both simulation study and real data analysis, it is suggested that one epoch is insufficient to provide good performance as it gives large mean squared error (MSE) and mean absolute error (MAE) values. Thus, the results support the conventional research that one epoch is insufficient to update the weights and give the results that are not so satisfactory. Specifically, analysis of simulated data consistently suggests that 10 epochs offer optimal results, as most of the MSE and MAE results show that 10 epochs have better performance on the DANN model. However, the best number of epochs cannot be determined as some results suggest other epochs as the best number of epochs. The US airline passenger data, which represents low-frequency data, shows that 10 epochs are the optimal number of epochs, while the Delhi temperature data, which represents high-frequency data, prefers 100 epochs. Overall, most of the MSE and MAE results fall to small epochs such as 10 or 20, and this seems to support the research that highlighted the aim of obtaining a minimal root mean squared error (RMSE) value while minimising the number of epochs as MSE is directly proportional to RMSE (Hastomo et al., 2021). Besides, it proves that lower epochs can give higher accuracy besides speeding up the program execution. Therefore, by considering all perspectives, including computing time, resources and accuracy, 10 epochs are highly recommended from the result of this study.

Another hyperparameter of interest is the batch size, where the range being investigated in this study consists of batch sizes of 32 and 64. The range is determined as the batch size should be not smaller than one and not greater than the number of samples in the training dataset. Also, the batch size should be a power of two to maximise GPU processing capabilities, and it is suggested to start exploring the best batch size starting from 32 (Kantel and Castelli, 2020). The smallest training dataset is 100 in the US airline passenger data. Thus, the range to be investigated in this study is batch sizes of 32 and 64. The optimal batch size cannot be determined as distinct batch sizes are suggested for different sample sizes using the MSE and MAE results. For example, in the real data analysis, by using MSE as the metric to measure the accuracy of the forecast, the US airline

passenger data suggest the optimal batch size of 32. However, by using MAE to measure the accuracy of the forecasting results, a batch size of 64 is selected. For Delhi temperature data, both MSE and MAPE suggest that a batch size of 64 is the best. The results from this study have the same outcomes as the time series prediction of voice disorder detection using audio data and the Indian stock market, which found optimal batch sizes of 32 and 64 respectively (Vuppalapati et al., 2019; Yadav, 2020). Both batch sizes of 32 and 64 are optimal in different conditions, where large batch sizes offer steady convergence and acceptable test performance while small batch sizes perform better in stability and generalisation (Lin, 2022).

Furthermore, the accuracy of the DANN model in forecasting the low-frequency time series data with seasonal variations is compared to that of high-frequency time series data with seasonal variations. The results show that low-frequency data gives better simulation results while high-frequency data gives better results for real data applications. This suggests a close relationship between optimal frequency range and seasonal data characteristics needs to be customized in DANN across diverse time series landscapes. In conclusion, the optimal values of batch size and the number of epochs is inconsistent for time series data with seasonal variations by determining through MSE and MAE values, as it depends on the patterns of the seasonal variations in the time series data. Different models of SARIMA and different coefficients implemented will make the time series data vary in seasonal variations and complexity, making the optimal values of batch size and the number of epochs keep changing to get a better result. Thus, it is crucial to perform hyperparameter tuning at the beginning of the learning process to ensure the selection of appropriate hyperparameter values, which significantly impact the learning outcome of a DNN model, leading to improved forecast accuracy results. The range of work that can be conducted with neural networks is limitless. Other hyperparameters can be considered in the future. For example, in this research, a DANN, which can be considered as two hidden layers of Multilayer Perceptron (MLP), is studied. The study can be extended by increasing the number of layers of MLP. Moreover, the study can also be extended by exploring other activation functions rather than ReLU, or more neurons in each layer. Learning rate and optimiser algorithm can be a new direction of study in the future too. Besides, other well-known neural networks such as RNN, LSTM and BiLSTM can be explored in future studies. Moreover, there are also some hybrid models, such as ARIMA-ANN, that can be included in the future study to investigate their performance on time series data with seasonal variation.

ACKNOWLEDGEMENTS

The authors are grateful to Universiti Putra Malaysia for supporting this project.

REFERENCES

Aslam, Z., Javaid, N., Aidil, M., Ijaz, M. T., ur Rahman, A., & Ahmed, M. (2020). An enhanced convolutional neural network model based on weather parameters for short-term electricity supply and demand. In L. Barolli, F. Amato, F. Moscato, T. Enokido & M. Takizawa (Eds.), *Advanced Information Networking and Applications: Proceedings of the 34th International Conference on Advanced Information Networking and Applications (AINA-2020)*, (pp. 22-35). Springer.

Azzeh, M. (2017). Online reputation model using moving window. *International Journal of Advanced Computer Science and Applications*., ***8*(4)**: 508-512.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems.*, ***2*(4)**: 303-314.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies., **1***: 4171–4186.

Faraway, J. & Chatfield, C. (1998). Time series forecasting with neural networks: a comparative study using the air line data. *Journal of the Royal Statistical Society Series C: Applied Statistics.*, ***47*(2)**: 231-250.

Gasparin, A., Lukovic, S., & Alippi, C. (2022). Deep learning for time series forecasting: The electric load case. *CAAI Transactions on Intelligence Technology.*, ***7*(1)**: 1-25.

Gikungu, S. W., Waititu, A., & Kihoro, J. (2015). Modeling inflation in Kenya: Comparison of SARIMA and generalized least squares models. *Mathematical Theory and Modeling.*, ***5*(12)**: 67-73.

Golingay, S. M. D. (2020). Out-of-sample forecasting of the Region XII, Philippines' non-metallics production volume using different modeling techniques. *International Journal of Scientific and Research Publications.*, ***10*(8)**: 541-553.

Guliyev, N. J., & Ismailov, V. E. (2018). On the approximation by single hidden layer feedforward neural networks with fixed weights. *Neural Networks.*, ***98***:296-304.

Hastomo, W., Karno, A. S. B., Kalbuana, N., Meiriki, A. & Sutarno (2021). Characteristic parameters of epoch deep learning to predict Covid-19 data in Indonesia. *Journal of Physics: Conference Series.*, ***1933*(1)**: Article 12050.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks.*, ***2*(5)**: 359-366.

Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express.*, ***6*(4)**: 312-315.

Khashei, M., & Hajirahimi, Z. (2019). A comparative study of series arima/mlp hybrid models for stock price forecasting. *Communications in Statistics-Simulation and Computation.*, ***48*(9)**: 2625-2640.

Kim, S., & Kim, H. (2016). A new metric of absolute percentage error for intermittent demand forecasts, *International Journal of Forecasting.*, ***32*(3)**: 669-679.

Kingma, D. P., & Ba, J. (2015). ADAM: A method for stochastic optimization. In K. Simonyan & A. Zisserman (Eds.), *3rd International Conference on Learning Representations (ICLR 2015)*, (pp. 1–15). International Conference on Learning Representations.

Kotur, D., & Zărkovĭc, M. (2016). Neural network models for electricity prices and loads short and long-term prediction. In A. Nikolic, K. Busawon, A. Maheri & G. Jankes (Eds.), *2016 4th International Symposium on Environmental Friendly Energies and Applications (EFEA)*, (pp. 1-5). IEEE.

Kouassi, K.H., & Moodley, D. (2020). An Analysis of Deep Neural Networks for Predicting Trends in Time Series Data. In A. Gerber (Ed.), *Artificial Intelligence Research. SACAIR 2021: Communications in Computer and Information Science*, ***1342*** (pp.119-140). Springer.

Lin, R. (2022). Analysis on the selection of the appropriate batch size in CNN neural network. *2022 International Conference on Machine Learning and Knowledge Engineering (MLKE)*, 106-109.

Lin, W. H., Wang, P., Chao, K. M., Lin, H. C., Yang, Z. Y., & Lai, Y. H. (2021). Wind power forecasting with deep learning networks: Time-series forecasting. *Applied Sciences.*, ***11*****(21)**: 10335.

Mahmood, A., Bennamoun, M., An, S., Sohel, F., Boussaid, F., Hovey, R., Kendrick, G., & Fisher, R. B. (2017). Deep learning for coral classification. In P. Samui, S. S. Roy, V. E. Balas (Eds.), *Handbook of Neural Computation*, (pp. 383-401). Elsevier.

Musbah, H., Aly, H. H., & Little, T. A. (2023). A proposed novel adaptive DC technique for non-stationary data removal, *Heliyon*, ***9*****(3)**, Article e13903.

Nelson, M., Hill, T., Remus, W., & O'Connor, M. (1999). Time series forecasting using neural networks: Should the data be deseason first? *Journal of Forecasting*, ***18*****(5)**, 359-367.

Ngoc, T. T., Dai, L. V., & Phuc, D. T. (2021). Grid search of multilayer perceptron based on the walk-forward validation methodology. *International Journal of Electrical and Computer Engineering*, ***11*****(2)**, 1742-1751.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Nwankpa, C. E. (2020). Advances in optimisation algorithms and techniques for deep learning. *Advances in Science, Technology and Engineering Systems Journal*, ***5*****(5)**, 563-577.

Picard, R. R., & Berk, K. N. (1990). Data splitting. *American Statistician*, ***44*****(2)**, 140–147.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

Sonoda, S., & Murata, N. (2017). Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, ***43*****(2)**, 233-268.

Smith, S. L., Elsen, E., & De, S. (2020). On the generalization benefit of noise in stochastic gradient descent. In H. Daumé III & A. Singh (Eds.), *ICML'20: Proceedings of the 37th International Conference on Machine Learning*, (pp. 9058-9067). Journal of Machine Learning Research.

Thomas, A., Walters, S., Gheytassi, M. M., Morgan, R., & Petridis, M. (2016). On the optimal node ratio between hidden layers: a probabilistic study. *International Journal of Machine Learning and Computing*, ***6*****(5)**, 241-247.

Thomas, A. J., Petridis, M., Walters, S. D., Gheytassi, S. M., & Morgan, R. E. (2017). Two hidden layers are usually better than one. In G. Boracchi, L. Iliadis, C. Jayne & A. Likas (Eds.), *Engineering Applications of Neural Networks: 18th International Conference, EANN 2017, Athens, Greece, August 25–27, 2017, Proceedings* (pp. 279-290). Springer.

Vuppalapati, J. S., Kedari, S., Kedari, S., Ilapakurti, A., & Vuppalapati, C. (2019). Artificial intelligent (AI) clinical edge for voice disorder detection. In Bi, Y., Bhatia, R., Kapoor, S. (Eds.), *Intelligent Systems and Applications: Proceedings of the 2019 Intelligent Systems Conference (IntelliSys)*, ***1038***, Springer.

Yadav, A., Jha, C., Sharan, A. (2020). Optimizing LSTM for time series prediction in Indian stock market. *Procedia Computer Science*, ***167***, 2091-2100.

Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing,* ***50***, 159-175.

Zhang, G. P., & Qi, M. (2005). Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, ***160*****(2)**, 501-514.

Ziari, H., Sobhani, J., Ayoubinejad, J., and Hartmann, T. (2016). Prediction of IRI in short and long terms for flexible pavements: ANN and GMHD methods. *International Journal of Pavement Engineering*, ***17*****(9)**, 776–788.

Zupan, J. (1994). Introduction to Artificial Neural Network (ANN) Methods: What They Are and How to Use Them. *Acta Chimica Slovenica*, ***41*****(3)**, 327–352.