Open Access Library Journal
2024, Volume 11, e11498
ISSN Online: 2333-9721
ISSN Print: 2333-9705

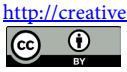
Development of a Simulated Artificial Neural Network Model for Stock Prediction

Njideka Nkemdilim Mbeledogu, Roseline Uzoamaka Paul

Department of Computer Science, Nnamdi Azikiwe University, Awka, Nigeria
Email: nn.mbeledogu@unizik.edu.ng

How to cite this paper: Mbeledogu, N.N. and Paul, R.U. (2024) Development of a Simulated Artificial Neural Network Model for Stock Prediction. *Open Access Library Journal*, **11**: e11498.
https://doi.org/10.4236/oalib.1111498

Received: March 28, 2024
Accepted: May 28, 2024
Published: May 31, 2024



Abstract

Artificial Neural Network (ANN) provides the learning ability for solving complex problems in similitude to the human brain. The research gives a detailed descriptive design of a simulated ANN. It expounded the architectural attributes (the network structure) showing the number and topology of the neurons with their interconnectivity and the neuro-dynamic attributes by employing the Levenberg Marquardt back propagation for the training and Gradient descent for the learning (adjusting the individual weight of the connection links). The Simulink in MATLAB was used to simulate the model and implemented with the stock data. The optimum performance for the number of hidden neurons was tested for 1000 epochs and the (4-50-5-1) structure had the least MSE training time of 1.4%. The performance of the model was evaluated using RMSE and it returned an error rate of approximately 0.0004. This shows the capability of the simulated ANN model in prediction.

Subject Areas

Artificial Intelligence

Keywords

Artificial Neural Network, Levenberg Marquardt Back Propagation, Gradient Descent

1. Introduction

Artificial Intelligence (AI) is the art and science of developing intelligent machines. These intelligent machines can be developed using hard computing or soft computing methods. Though both of them aim to achieve general intelligence as their goal, their approaches to adaptation to many situations are different. Hard computing huge set of conventional methods such as stochastic and

statistical methods uses crisp binary-based computation that needs an exact input sample but Soft Computing methods (Artificial Neural Network, Fuzzy Logic, Genetic Algorithm and Probabilistic Reasoning) are founded on the importance given to issues like learning, cognition, precision, certainty, recognition, rigor, approximate reasoning and functional approximation and randomized search to solve non-linear and mathematically un-modeled system problem, thus, deals with ambiguous and noisy data [1].

Artificial Neural Network (ANN) provides the learning ability for solving complex problems in similitude to the human brain [2]. It is a massively parallel distributed processor that has a natural capability for storing experiential knowledge [3]. It could also be said to be basically mathematical models of information processing that provide a method of representing relationships that is quite different from that of turing machines or computers with stored programs [4].

Due to its benefits, researchers have delved into ANN for prediction. Some of the works are:

[5] predicted students' performance with ANN using demographic traits such as age, gender etc. The study aimed at selecting students with high prediction of success for admission using previous academic records. The research did not expound on the necessary features of the ANN applied. These features include the high level model of the ANN design which involves the type of machine learning architecture deployed, the architectural attributes (number and topology of network) and the neuro-dynamic attributes (functionality) to show the training and the learning algorithm.

[6] conducted a comprehensive literature survey on using ANN for building energy prediction. The researchers showed the progressive rate of the research trend on ANNs in building energy prediction in the last five years by introducing in detail twelve ANN architectures applied and discussing the challenges encountered for future investigation.

[7] worked on stock price prediction using an artificial neural network integrated moving average. The research focused on improving the quality of the experimental data using an integrated moving average for the ANN to train on. Both the structure and the functionality of the ANN were not properly analyzed in detail.

Based on these, the aim of this research is to give a well detail descriptive design of a simulated ANN showing the architectural attributes and the neuro-dynamic attributes that will guide researchers in properly developing their desired ANN model.

2. Literature Review

The human brain (biological neural network) consists of a complex set of interconnected neurons, and their interaction produces characteristics associated with intelligence. The development of ANN is based on the information processing that occurs in the simplest unit of the brain called neuron [8]. A biological neuron has three main components: the soma (cell body), the axon and dendrites. The

dendrites accept impulses as inputs from other neuron for the soma to process (learning) through the help of synapses that enables the transmission of these impulses from the dendrites to the cell body. The axon then transfers the processed signals or impulses to the other neurons. The firing rate of the neuron (activation) happens when the impulses are strong enough to reach the threshold.

Model of a Neuron

A typical neuron model has three basic elements (Figure 1). They are:

1) A set of synapses or connecting links, each is delineated by a strength (weight) of its own. A signal input_n at the input of synapse n connected to a neuron k is multiplied by the synaptic weight w_{kn}. If the weight is positive then the associated synapse is excitatory else, it is inhibitory.

2) A linear combiner that sums the input signals, their respective weighted synapses and a bias $b = 1$.

3) An activation function for limiting the amplitude of the output of a neuron with normalized interval range of the output of a neuron as [0, 1] or [−1, 1].

$$Y = f\left(\sum_{i=1}^{n} x_i w_{ki} + b\right) \tag{1}$$

The learning process of the neuron is achieved by adjusting the weights of the interconnections according to some applied learning algorithms. The manner in which the neurons of a neural network are arranged is esoterically linked with the learning algorithm used to train the network. Basically, there are two learning paradigms [9]. They are Supervised learning and Unsupervised learning.

When the network is provided with a correct answer (output) for every input pattern and the weights are determined to allow the network to produce answers as close as possible to the known correct answers, it is referred to as Supervised Learning [10]. Examples are Single Layer Perceptron, Multiple Layer Perceptron (MLP) and Radial basis neural network (RBNN). In contrast to this, when the

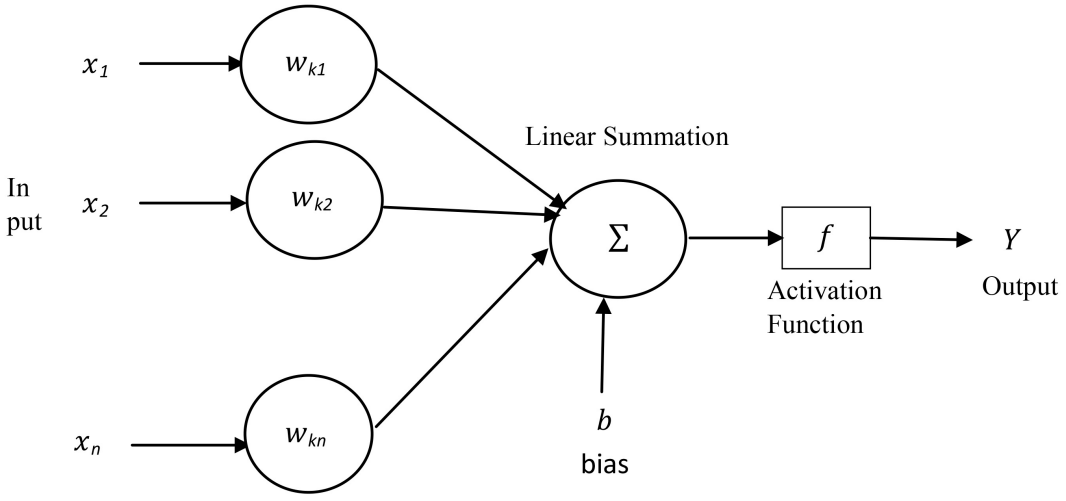


Figure 1. Model of a neuron.

system surveys the underlying structure in the data by organizing the patterns into categories from their correlations, it applies an unsupervised mechanism. An example is the Self Organizing Map (SOM) such as *Kohonen SOM*.

These learning paradigms gave rise to different types of neural network architectures. Some of the ANN architectures are Multi-layer Perceptron (MLP), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Generative Adversarial Networks (GAN).

The basic attributes of ANNs can be classified into Architectural attributes and neuro-dynamic attributes [11]. The architectural attributes define the network structure, that is, number and topology of neurons and their interconnectivity. The neuro-dynamic attributes define the functionality of the ANN.

3. Materials and Methods

3.1. Data Collection

The daily price lists from http://www.cashcraft.com/ for 2012 was collected as the research experimental stock data for Julius Berger (Construction Company) and GlaxoSmith Kline in 2016 as shown on Table 1 and Table 2. Four independent stock data (High, Low, Close and Open Price) were used as the input data (technical factors).

Table 1. Sample of Julius Berger experimental stock data.

Days	Date	Open Price	High Price	Low Price	Close Price
1	3/1/2012	31.6	31.6	31.6	31.6
2	4/1/2012	31.6	31.6	31.6	31.6
3	5/1/2012	31.61	31.6	31.6	31.6
4	6/1/2012	31.6	31.6	31.6	31.6
5	17/1/2012	33.18	33.18	33.18	31.6
6	18/1/12	33.18	33.18	33.18	33.18
7	19/1/12	32.05	32.05	32.05	33.18
8	20/1/12	32.05	32.05	32.05	32.05
9	24/1/12	30.7	30.7	30.7	32.05
10	25/1/12	30.7	30.7	30.7	30.7
11	26/1/12	31	31	31	30.7
12	27/1/12	31.5	31.5	31.5	31
13	30/1/12	31.5	31.5	31.5	31.5
14	31/1/12	40	40	31.5	31.5
15	1/2/2012	31.5	31.5	31.5	31.5
16	2/2/2012	30.01	30.01	30	31.5
17	3/2/2012	30	30	29.11	30
18	7/2/2012	29.11	29.11	29.11	29.11
19	8/2/2012	29.2	29.2	28	29.11
20	9/2/2012	27	27	27	28

Table 2. Sample of GlaxoSmithKline Consumer Plc stock data.

Days	Date	Open Price	High Price	Low Price	Close Price
1	3/1/2012	23	23	23	23
2	4/1/2012	23	23	23	23
3	5/1/2012	23	23	23	23
4	6/1/2012	22.85	23	22.85	23
5	10/1/2012	21.85	21.85	21.85	23
6	11/1/2012	21.85	21.85	21.85	21.85
7	12/1/2012	22.9	22.9	22.9	21.85
8	16/1/12	23	23	23	22.9
9	17/1/12	23	23	23	23
10	18/1/12	23	23	23	23
11	19/1/12	23	23	23	23
12	24/1/12	23	23	23	23

3.2. Simulation

The Simulink in MATLAB was used to simulate the model. MATLAB serves as a vehicle for modeling dynamic systems while the Simulink provides a graphical user interface (GUI) that is used in building block diagrams, performing simulations, as well as analyzing results [12].

3.3. ANN Modeling

The ANN maps the input and the associated parameters as shown in Figure 2.

In order for the network to deal with more complex non-linear problems, hidden non-linear layers were added to form a multilayer perceptron. A Multi-layer perceptron model (MLP) architecture was used because it can be trained to approximate most functions arbitrarily well.

Four layered MLP was randomly selected. The MLP comprised of four layers – input layer, two hidden layers and the output layer as shown in Figure 3.

The MLP model is a feed forward neural network model trained with back propagation algorithm. The MLP is structured in a feed forward topology whereby each unit gets its input from the previous one. Network trained with back propagation algorithm are networks with feedback connections that is global in nature.

The basic parameters needed for the design of an MLP ANN Model were presented in Table 3.

3.4. Transfer Function (Activation Function)

Log-sigmoid and Purelin activation functions were deployed within the layers because they can be trained to approximate most functions arbitrarily well [13]. The financial time series under consideration is highly non-linear and requires a

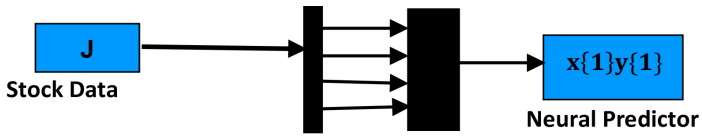


Figure 2. Mapping of the input parameters into the neural predictor.

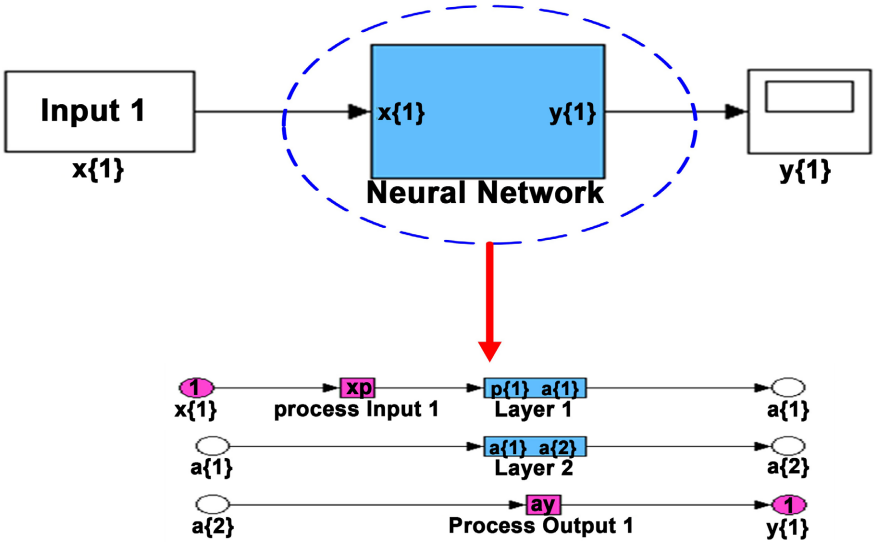


Figure 3. ANN block showing the multi-layers.

Table 3. Basic parameters in designing MLP model.

Type of input data - Technical factors
Training - Learning rate - Momentum term - Training tolerance - Epoch size - Goal - Learning rate limit - Number of times to randomize weights - Size of training, testing and validation sets
Topology - Number of input neurons - Number of hidden Layers - Number of hidden neurons in each layer - Number of output neurons - Transfer function for each neuron - Error function

sufficiently non-linear function to represent all the properties of the series. The logistic function of log-sigmoid depicted in Figure 4 was chosen for the hidden layers because it converges faster to a solution, therefore reduces the cost of computation for the hidden layer with multiple nodes. This transfer function

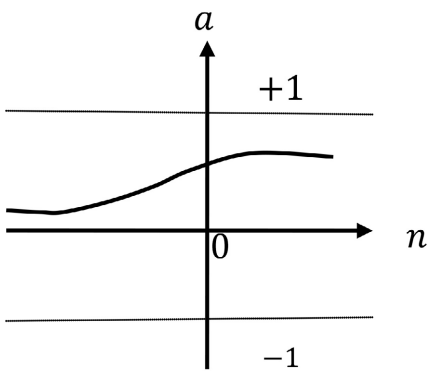


Figure 4. Log-sigmoid transfer function.

takes the input (which may have any value between plus and minus infinity) and squashes the output into the range 0 to 1.

$$a = logsig(n) \tag{2}$$

Purelin activation function [14] shown in **Figure 5** is also known as Linear transfer function. It does not change the summation results but transfers them after the summation process, hence, the outputs have no limit.

$$a = Purelin(n) \tag{3}$$

3.5. Time Delay Layer (TDL)

TDL is an important aspect of the interlinking of layers. A time-delay layer was incorporated into the input layer of a static multi-layer forecasting system in which the dynamics appear only as shown in **Figure 6**. The response of the neural network in time t is based on the inputs in times $(t-1),(t-2),\cdots,(t-n)$. A mapping performed by the TDL produces a $y(k)$ output at time k as:

$$y(K) = f\left(u(k);u(k-1),\cdots,u(k-m)\right) \tag{4}$$

where $u(k)$ is the stock input at time k and m is the maximum adapted time-delay.

3.6. Neuro-Dynamic Attributes

Training and learning functions are mathematical procedures used to automatically adjust the network's weights and biases. The training function dictates a global algorithm that affects all the weights and biases of a given network while the learning function can be applied to individual weights and biases within a network [15].

The system deployed a supervised paradigm by adopting Levenberg Marquardt (LM) back propagation for training and Gradient Descent (GD) for learning.

LM back propagation algorithm was implemented using the trainlm function. It was used because it works extremely well in practice, considered to be the most efficient algorithm and has a high speed in convergence [16]. It is a variation

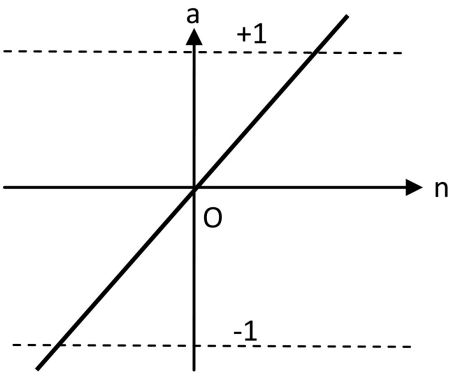


Figure 5. Purelin transfer function.

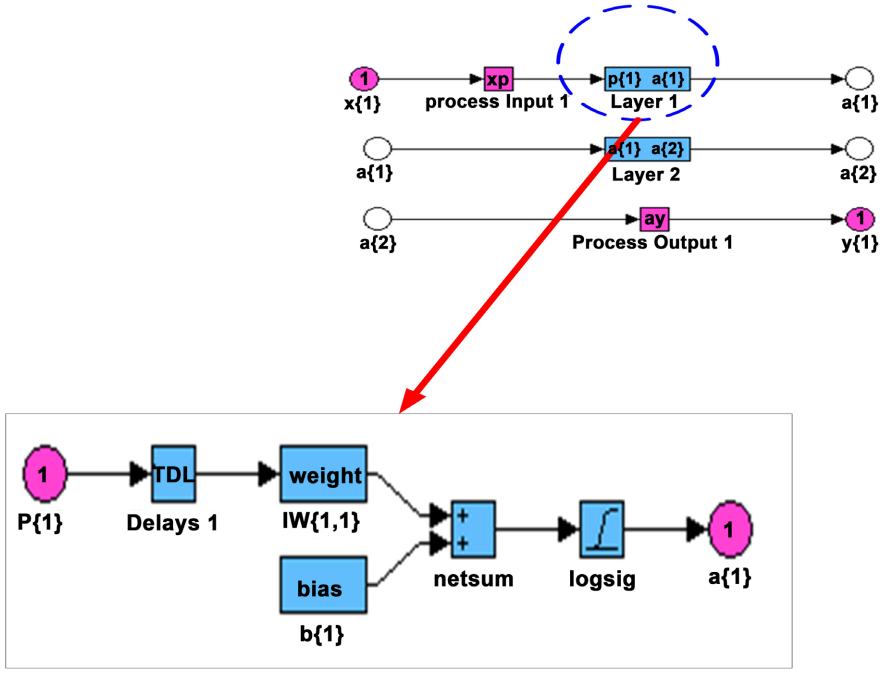


Figure 6. Internal structure of a layer showing the time delay, weights, activation function and bias.

of Newton's method that was designed for minimizing functions that are sum of squares of other non-linear functions. This is very well suited to neural network training where the performance index is the mean squared error [13]. GD back propagation was because of its popularity.

3.6.1. Training Algorithm

LM back propagation algorithm is given:

Step 1: The independent sample stock data are the inputs to the neural network. Propagate the input forward through the network by selecting random weights and biases:

$$a^{m+1} = f^{m+1}\left(W^{m+1}a^{m} + b^{m+1}\right) \quad \text{for} \quad m = 0,1,\cdots,M-1 \tag{5}$$

Step 2: Calculate the errors

$$e_q = t_q - a_q^M \tag{6}$$

where e_q is the error, t_q is the target value and a_q^M is the output.

Step 3: Compute the sum of squared errors over all inputs, $F(x)$:

$$F(x) = \sum_{q=1}^{Q} \left(t_q - a_q\right)^{\mathrm{T}} \left(t_q - a_q\right) \tag{7}$$

$$= \sum_{q=1}^{Q} e_q^{\mathrm{T}} e_q$$

$$= \sum_{q=1}^{Q} \sum_{j=1}^{s^m} \left(e_{j,q}\right)^2$$

$$= \sum_{i=1}^{N} \left(v_i\right)^2 \tag{8}$$

where $e_{j,q}$ is the jth element of the error for the qth input/target pair and v is the error vector.

$$v^{\mathrm{T}} = \begin{bmatrix} v_1 & v_2 & \cdots & v_N \end{bmatrix} \tag{9}$$

$$= \begin{bmatrix} e_{1,1} & e_{2,1} & \cdots & e_{s^m,1} & e_{1,2} & \cdots & e_{s^m,Q} \end{bmatrix} \tag{10}$$

Step 4: Compute the jacobian matrix $J(x)$:

$$J(x) = \begin{bmatrix} \dfrac{\partial e_{1,1}}{\partial w_{1,1}^1} & \cdots & \dfrac{\partial e_{1,1}}{\partial b_1^1} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial e_{s^m,1}}{\partial w_{1,1}^1} & \cdots & \dfrac{\partial e_{s^m,1}}{\partial b_1^1} \end{bmatrix} \tag{11}$$

Step 5: Calculate the sensitivity with the recurrence relations

Initialize the back propagation with

$$S_q^{-m} = -F^m\left(n_q^m\right) \tag{12}$$

Step 6: Propagate each column of the matrix S_q^{-m}

$$S_q^{-m} = F^m\left(n_q^m\right)\left(W^{m+1}\right)^{\mathrm{T}} S_q^{-m+1} \tag{13}$$

Step 7: Augment the individual matrices into marquardt sensitivities:

$$S^{-m} = \left\langle S_1^{-m} \middle| \cdots \middle| S_Q^{-m} \right\rangle \tag{14}$$

Note: For each input presented to the network, the sensitivity vectors are propagated back. This is because the derivatives of each individual error is computed and not the derivative of the sum of squares of the errors. For every input applied to the network there will be S^m errors (one for each element of the network output). For each error, there will be one row of the Jacobian matrix.

Step 8: Compute the elements of the Jacobian matrix:

$$\left[J\right]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w_{i,j}^m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \times \frac{\partial n_{i,q}^m}{\partial w_{i,j}^m} = s_{i,h}^{-m} \times \frac{\partial n_{i,q}^m}{\partial w_{i,j}^m} = s_{i,h}^{-m} \times a_{j,q}^{m-1} \tag{15}$$

$$\left[J\right]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b_i^m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \times \frac{\partial^m_{n_{i,q}}}{\partial b_i^m} = s_{i,h}^{-m} \times \frac{\partial n_{i,q}^m}{\partial b_i^m} = s_{i,h}^{-m} \tag{16}$$

if x_l is a bias.

Step 9: Solve to obtain Δx_k

$$\Delta x_k = -\left[J^{\mathrm{T}}(x_k)J(x_k) + \mu_k I\right]^{-1} J^{\mathrm{T}}(x_k)v(x_k) \tag{17}$$

Step 10: Re-compute the sum of squares errors use $x_k + \Delta x_k$. If this new sum of squares is smaller than computed in step 1, then divide μ by v, let

$$x_{k+1} = x_k + \Delta x_k \tag{18}$$

And go back to step 1. If the sum of squares is not reduced, then multiply μ by v.

And go back to step 9.

The algorithm is assumed to have converged when the norm of the gradient is

$$\nabla F(x) = 2J^{\mathrm{T}}(x)v(x) \tag{19}$$

less than some predetermined value, or when the sum of squares has been reduced to some error goal.

3.6.2. Learning Algorithm

Gradient descent (GD) learning algorithm with learning rate and momentum coefficient was used for the learning. It was implemented using the Learngdm Function.

GD algorithm [17] is given as:

Step 1: Having the network:

$$a_{net,j} = \left(\sum_{i=1}^{l} w_{ij}o_i\right) + \theta_j \tag{20}$$

where o_j is the output of the j^{th} unit, o_i is output of the i^{th} unit, w_{ij} is the weight of the link from unit i to unit j, $a_{net,j}$ is the net input activation function for the j^{th} unit, θ_j is the bias for the j^{th} unit and c_j is the gain of the activation function.

Step 2: The gain update expression for a gradient descent (GD) method is calculated by differentiating the following error term E with respect to the corresponding gain parameter.

The network error is defined as

$$E = \frac{1}{2}\sum\left(t_k - o_k\left(o_j, c_k\right)\right)^2 \tag{21}$$

Step 3: Calculate $\dfrac{\partial E}{\partial c_k}$ for the output unit and $\dfrac{\partial E}{\partial c_j}$ for the hidden units.

The respective gain values would then be updated with the following equations:

$$\Delta c_k = \eta\left(-\frac{\partial E}{\partial c_k}\right) \tag{22}$$

$$\Delta c_j = \eta\left(-\frac{\partial E}{\partial c_j}\right) \tag{23}$$

$$\frac{\partial E}{\partial c_k} = -\left(t_k - o_k\right) o_k \left(1 - o_k\right)\left(\sum w_{jk} o_j + \theta_k\right) \tag{24}$$

Therefore, the gain update expression for links connecting to output nodes is:

$$\Delta c_k \left(n+1\right) = \eta\left(t_k - o_k\right) o_k \left(1 - o_k\right)\left(\sum w_{jk} o_j + \theta_k\right) \tag{25}$$

$$\frac{\partial E}{\partial c_j} = \left[-\sum_k c_k w_{jk} o_k \left(1 - o_k\right)\left(t_k - o_k\right)\right] o_j \left(1 - o_j\right)\left(\sum_j w_{ij} o_i + \theta_j\right) \tag{26}$$

Therefore, the gain update expression for the links connecting hidden nodes is:

$$\Delta c_j \left(n+1\right) = \eta\left[-\sum_k c_k w_{jk} o_k \left(1 - o_k\right)\left(t_k - o_k\right)\right] o_j \left(1 - o_j\right)\left(\sum_j w_{ij} o_i + \theta_j\right) \tag{27}$$

Step 4: Similarly, the weight and bias expressions are calculated as follows:

The weights update expression for the links connecting to output nodes:

$$\Delta w_{jk} \left(n+1\right) = \eta\left(t_k - o_k\right) o_k \left(1 - o_k\right) c_k o_j + \alpha \Delta w_{jk} \left(n\right) \tag{28}$$

where the LR, η and MC, α are randomly generated.

Step 5: Similarly, the bias update expressions for the output nodes would be:

$$\Delta \theta_k \left(n+1\right) = \eta\left(t_k - o_k\right) o_k \left(1 - o_k\right) c_k + \alpha \Delta w_{jk} \left(n\right) \tag{29}$$

The weight update expression for the links connecting to hidden nodes is:

$$\Delta w_{jk} \left(n+1\right) = \eta\left[\sum_k c_k w_{jk} o_k \left(1 - o_k\right)\left(t_k - o_k\right)\right] c_j o_j \left(1 - o_j\right) o_i + \alpha \Delta w_{ik} \left(n\right) \tag{30}$$

Step 6: Similarly, the bias update expressions for the hidden nodes would be:

$$\Delta \theta_j \left(n+1\right) = \eta\left[\sum_k c_k w_{jk} o_k \left(1 - o_k\right)\left(t_k - o_k\right)\right] c_j o_j \left(1 - o_j\right) + \alpha \Delta w_{ik} \left(n\right) \tag{31}$$

3.6.3. Training Parameters

The training parameters used in the ANN block are shown in Table 4. For each of the parameter, an initial value was selected and constantly adjusted until the desired performance was achieved. Two approaches were considered for convergence. They were the learning rate and the momentum coefficient.

1) Learning Rate (LR): This is one of the most effective means to accelerate the convergence of Back Propagation (BP) learning. The value is usually set to be

Table 4. Training parameters.

Parameter	Value	Description
net.trainParam.epochs	1000	Maximum number of epochs (cycles)
net.trainParam.goal	1e−5	Performance goal in terms of mean square error
net.trainParam.Ir	0.05	Learning rate
net.trainParam.mc	0.5	Momentum Coefficient
net.trainParam.max_fail	100	Maximum validation value
net.trainParam.mem_reduc	2	Memory requirement specification
net.trainParam.min_grad	1e−50	Minimum Performance Gradient
net.trainParam.time	1000	Maximum training time

constant which means that the selected value is employed for all weights in the whole learning process. Taking into consideration that the algorithm will take longer time to converge or may never converge if the LR is too small and may oscillate if it is too high, a value was initially randomly selected and adjusted until the desired performance was obtained.

2) Momemtum Coefficient (MC): Without MC, the network can slide through shallow local minima. The value of the gain parameter, c_j, directly influences the slope of the activation function. For large gain values, ($c > 1$), the activation function approaches a "step function" whereas for small gain values ($0 < c \leq 1$), the output values change from zero to unity over a large range of the weighted sum of the input values and the sigmoid function approximates a linear function.

3.6.4. Training

The stock data series were first partitioned into three disjoint sets: the training set, the validation set, and the test set. 60% of the stock data was used for training, 20% for validation and 20% for testing. These stock data were stored in the MATLAB workspace with variable name.

The training and validation data sets were used during training. The training set is the data set used to adjust the weights on the neural network while the validation set is the data set used to minimize over-fitting. Any increase in accuracy over the training data set actually yields an increase in accuracy over a data set that has not been shown to the network before, or at least the network hasn't trained on it. If the accuracy over the training data set increases, but the accuracy over the validation data set stays the same or decreases, then one is over-fitting the neural network and should stop training. Also, training is stopped at the moment the validation error starts to rise.

Training Conditions

Training stops when any of the conditions below occurs:

- The maximum number of epochs (repetitions) is reached.
- The maximum amount of time is exceeded.
- Performance is minimized to the goal.
- The performance gradient falls below min_grad.
- Momentum (mu) exceeds mu_max.
- Validation performance has increased more than max_fail times since the last time it decreased.

3.7. Architectural Attributes

This defines the network structure, that is, number and topology of neurons and their interconnectivity.

1) Choice of Number of Neurons for Each Layer

The numbers of layers for the structure were randomly selected with the aim of determining the one with the least Mean Square Error (MSE) structure with most emphasis laid on the hidden layers. The input layer had a static number of

neurons to be four (4) because of the 4 independent input data (Open price, Low price, High price and Close price). The output layer is also restricted to one neuron because only one Closing price is expected as the predicted value.

For the hidden layers, two layered hidden structures with different number of neurons (various network structures) were considered randomly and tested for different epochs of training time up till 1000 epochs. Figure 7 shows the graphical representation of training for optimal performance for 30 epochs. Optimum performance experimentation was carried out with the aim of determining the number of hidden neurons in the hidden layers with the least MSE (Table 5).

2) The Mean Square Error (MSE)

This is an average of the squares of the difference between the actual observations and those predicted. It is a measure of how close a fitted line is to data points. It is calculated [18] as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(X_{obs,i} - X_{model,i}\right)^2 \tag{32}$$

where $X_{obs,i}$ is observed values and $X_{model,i}$ is modeled values at time.

a-b-c-d structure on Table 5 represents the number of neurons in the input layer-hidden layer1-hidden layer2-output layer. The MLP structure with 4-50-5-1

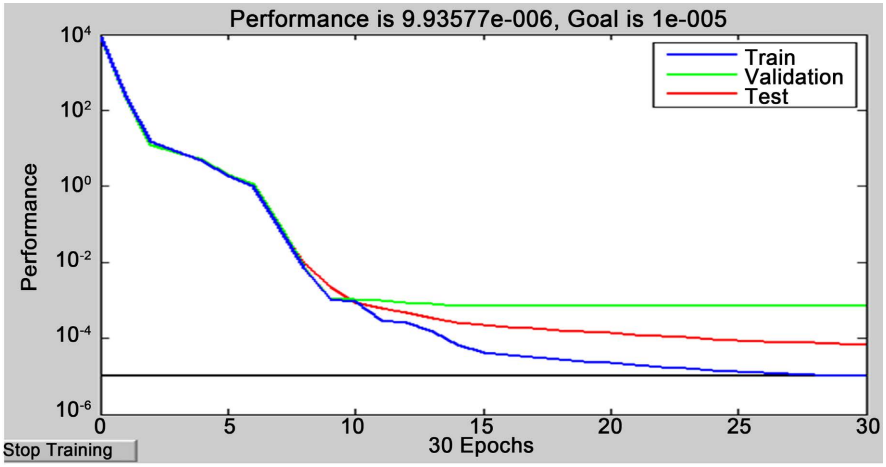


Figure 7. Graphical representation of training for optimal performance.

Table 5. Optimum performance for number of hidden neurons at 1000 epochs.

1000Epochs		
Number Neurons	**% Training Time**	**MSE**
4-10-5-1	1.9	9.92647e−006
4-20-5-1	3.2	9.861e−006
4-30-5-1	2.1	9.9529e−006
4-40-5-1	5.5	9.78957e−006
4-50-5-1	**1.4**	**7.66117e−006**
4-60-5-1	4.9	9.86415e−006

returned both least mean square error and training time, thus, became the optimal structure for prediction.

Figure 8 and Figure 9 show the first hidden layer having 50 neurons with log sigmoid activation function, and the second hidden layer having 5 neurons with log sigmoid function and the last layer which is the output process having one neuron with Purelin function respectively.

The internal structure of the connections between the hidden layer 2 and the last layer is shown in Figure 9. A Multiplexer was employed. A multiplexer (also called a data selector or mux) is a hardware device that accepts multiple inputs and allows only one to go through as an output [19].

Once a model is selected based on the validation set, the test set data is applied on the network model and the error for this set is determined.

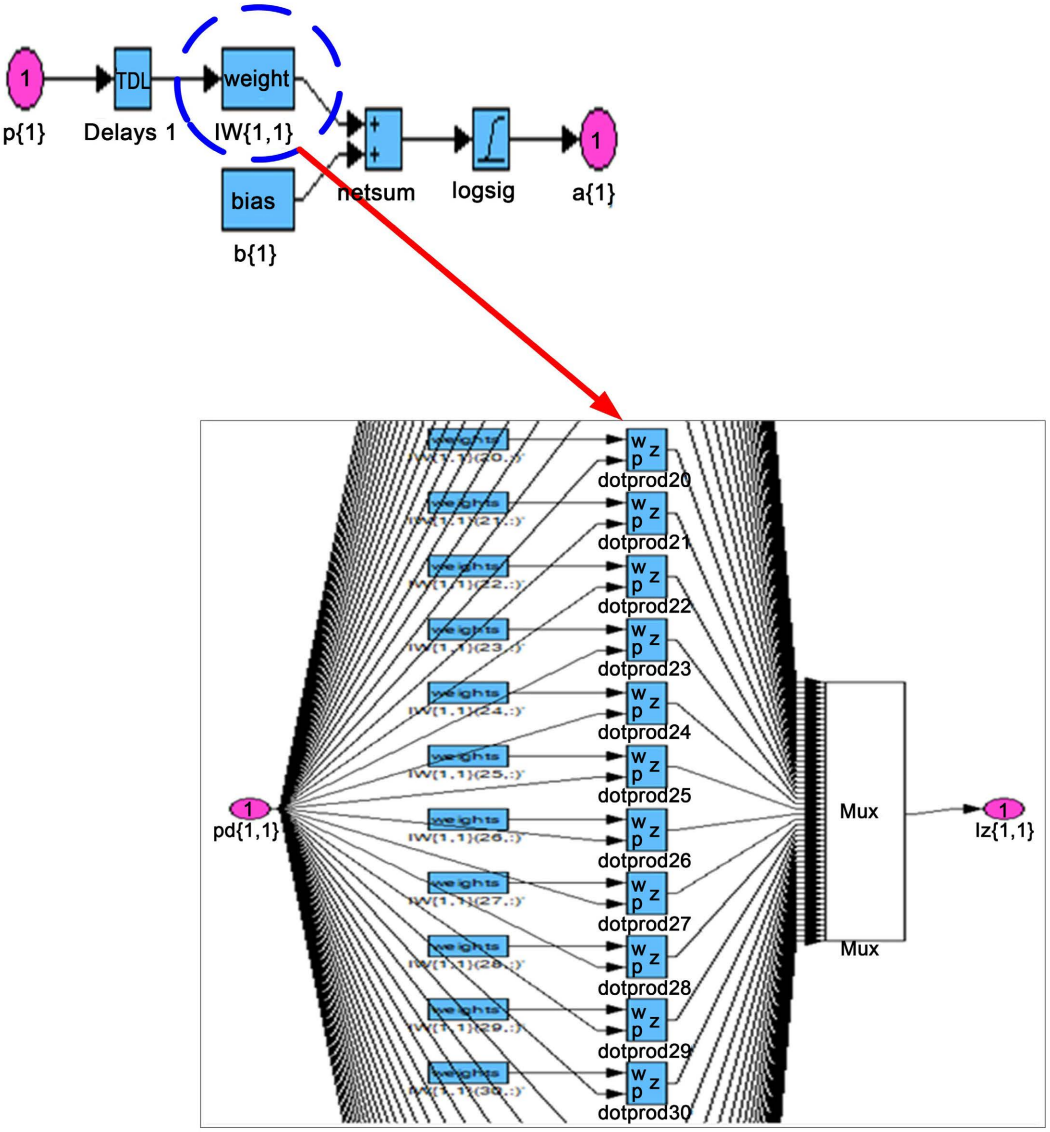


Figure 8. Simulink block diagram showing the weight structure of the first hidden layer having 50 neurons connecting to the input matrix of the second hidden layer.

3.8. Testing the Network

The network was tested for estimating the network's ability to generalize. The testing data set was used in order to confirm the actual predictive power of the network.

4. Results and Discussions

Functional tests and comparisons as presented on Table 6 and Table 7 were

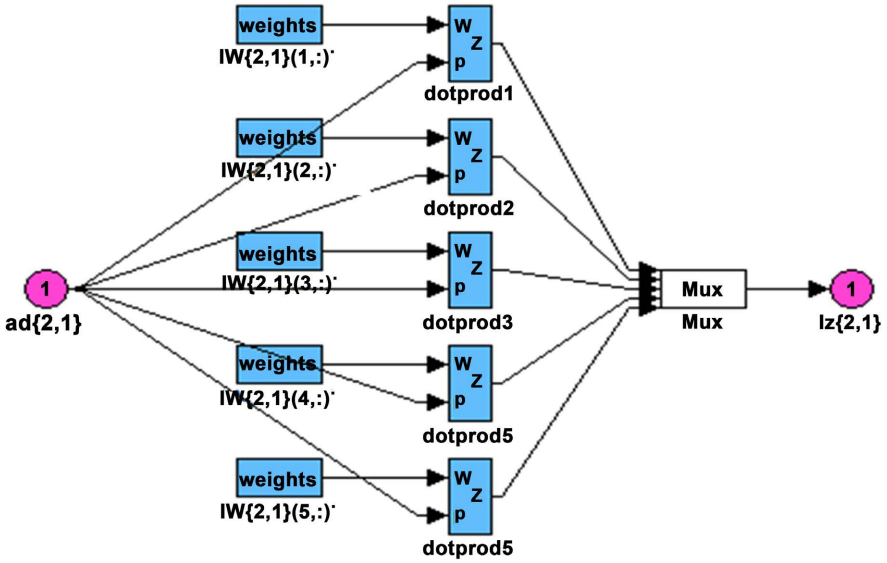


Figure 9. Simulink block diagram showing the weight structure of second hidden layer with 5 neurons.

Table 6. Comparison between actual stock price and ANN predictions for Julius Berger.

Actual Close Price	ANN Predictions
31.06	31.06012
31.06	31.06012
31.06	31.06012
31.06	31.06012
31.06	31.06027
32.61	32.61015
34.24	34.24013
34	34.00015
34	34.00015
34	33.99997
33.01	33.01014
30.61	30.61006
30.61	30.61006
30.61	30.61006

Table 7. Comparison actual stock price and ANN Predictions for GlaxoSmith Kline Plc.

Actual Close Price	ANN Predictions
21.85	22.99999
21.85	21.85
22.9	21.84999
23	22.9
23	23
23	23
23	23
22.7	22.70001
22.7	22.70001
22.7	22.70001
22.7	22.70001
22.7	22.70001
22.7	22.70001
22.7	22.70001
22.7	22.70001
22.7	22.70001

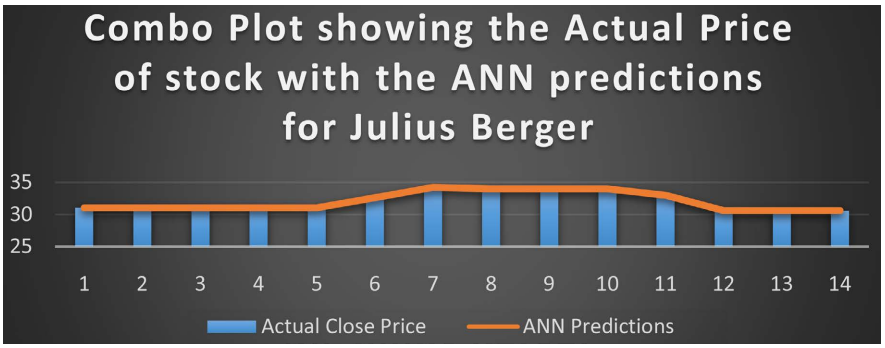


Figure 10. Combo plot of stock prediction for Julius Berger.

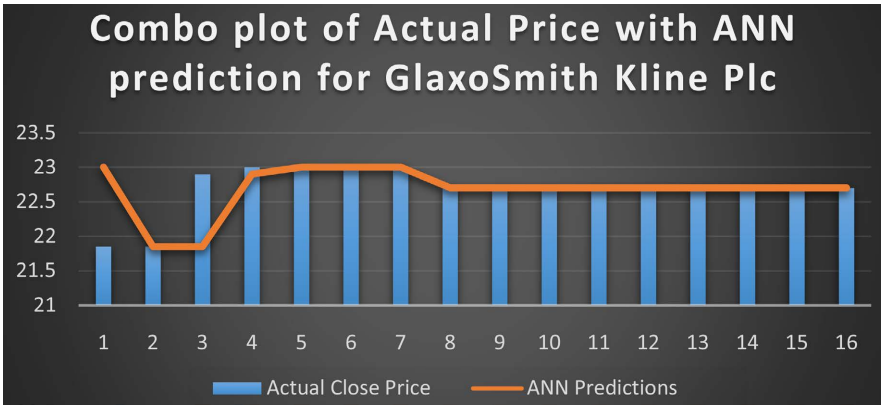


Figure 11. Combo plot of stock prediction for GlaxoSmith Kline Plc.

done to establish its ability to forecast stock price accurately. It can be observed that the ANN predictions were within the neighborhood of prediction and very close with the actual close price. Figure 10 and Figure 11 show the combo graphical representations of ANN predictions for Julius Berger and GlaxoSmith Kline Plc respectively.

Performance Evaluation

The predictive performance analysis of the model was calculated using Root Mean Square Error (RMSE) as shown in Equation (33). RMSE is also a notable predictive performance metric and can be calculated as

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}\left(X_{obs,i} - X_{model,i}\right)^2}{n}} \tag{33}$$

where $X_{obs,i}$ is observed values and $X_{model,i}$ is modeled values at time.

The predictive performance of the simulated ANN model returned very small error size of 0.0004. This signifies the accuracy of its prediction.

5. Conclusion

ANN provides experimental paradigm for parallel distributed processing and simulated models of it can be used to solve statistical analysis, data modeling and complex real life problems.

Conflicts of Interest

The authors declare no conflicts of interest.

References

[1] Huang, Y., Lan, Y., Thomson, S., Fang, A., Hoffmann, W. and Lacey, R. (2010) Development of Soft Computing and Applications in Agricultural and Biological Engineering. *Computers and Electronics in Agriculture*, **71**, 107-127. https://doi.org/10.1016/j.compag.2010.01.001

[2] Haag, S., Cummings, M. and Dawkins, J. (1998) Management Information System for the Information Age. Mc-Graw-Hill, USA, 526.

[3] Haykin, S. (1998) Neural Networks: A Comprehensive Foundation, Macmillan College Publishing Company, Inc., London, USA, 1-41.

[4] Fausett, L. (1996) Fundamentals of Neural Network: Architectures, Algorithms and Applications, Prentice Hall, Upper Saddle River, New Jersey, 1-14.

[5] Kehinde, A.J., Adeniyi, A.E., Ogundokun, R.O., Gupta, H. and Misra, S. (2022) Prediction of Students' Performance with Artificial Neural Network Using Demorgraphic Traits. In: Singh, P.K., Singh, Y., Chhabra, J.K., Illés, Z. and Verma, C., Eds., *Recent Innovation in Computing. Lecture Notes in Electrical Engineering*, Springer, Singapore, 613-624. https://doi.org/10.1007/978-981-16-8892-8_46

[6] Lu, C.J., Li, S.H. and Lu, Z.J. (2022) Building Energy Prediction Using Artificial Neural Network: A Literature Survey. *Energy and Building*, **262**, Article 111718. https://doi.org/10.1016/j.enbuild.2021.111718

[7] Suryani, I. and Buani, D.C.P. (2020) Stock Price Prediction Using Artificial Neural

Network Integrated Moving Average. *Journal of Physics*: *Conference Series*, **1641**, Article 012028. https://doi.org/10.1088/1742-6596/1641/1/012028

[8] Abhijit, S.P. and Macy, R. (2000) Pattern Recognition with Neural Networks in C++. CRC Press, Boca Raton, Florida, 1-43.

[9] Krose, B. and Van Der Smagt, P. (1996) An Introduction to Neural Networks. 8th Edition, University of Armsterdan, Amsterdam. http://prolland.free.fr/works/ai/docs/neuro-intro.pdf

[10] Samarasinghe, S. (2006) Neural Networks for Applied Sciences and Engineering from Fundamentals to Complex Pattern Recognition, Auerbach Publications, Taylor and Francis Company, New York. https://doi.org/10.1201/9781420013061

[11] Kartalopoulous, S.V. (1996) Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications, IEEE Press, Piscataway Township.

[12] Hill, J.M. (2001) Introduction to Simulink. https://www.slideserve.com/wilona/introduction-to-simulink

[13] Hagan, M.T., Demth, H.B. and Beale, M. (1996) Neural Network Design. PWS Publishing Company.

[14] Mathworks (2024) Purelin Linear Transfer Function. https://www.mathworks.com/help/deeplearning/ref/purelin.html

[15] Mathworks (2014) MATLAB Programming Fundamentals. https://pdfroom.com/books/matlab-programming-fundamentals/zydD8wQQd14

[16] Azhar, K.A. and Kunid, W. (2007) Daily Groundwater Level Fluctuation Forecasting Using Soft Computing Technique. *Journal of Nature and Science*, **5**.

[17] Rehman, M.Z. and Nawi, N.M. (2011) Improving the Accuracy of Gradient Descent back Propagation Algorithm (GDAM) on Classification Problems. *International Journal on New Computer Architectures and Their Applications*, **1**, 838-847.

[18] Vernier (2011). What Are Mean Squared Error and Root Mean Squared Error? http://www.vernier.com/til/1014/

[19] Jadoon, R. (2010) Multiplexer and De-Multiplexer. https://jadoon956.wordpress.com/wp-content/uploads/2010/10/multiplexer-demultiplexer.pdf