مجلة المثنى للعلوم الادارية والاقتصادية
ONLINE ISSN : 2572-5386 … PRINT ISSN : 2226-1419
website : www.muthjaes.net

Performance Comparison of Neural Networks (MLP, RBFNN, ERNN, JRNN) Models for Stock Prices Forecasting to Bank of Palestine

Shady I. Altelbany [a] , Anwar A. Abualhussein [b]
AL-Azhar University, Faculty of Economics and Administration Sciences

Abstract

This study aimed to Performance Comparison of Neural Networks (MLP, RBFNN, ERNN, JRNN) Models for the time series data of a monthly Stock Prices to Bank of Palestine from Nov. 2005 to Oct. 2020, and comparing between models to see which one is better in forecasting. The results of applying the methods were compared through the (MAPE, MAE, RMSE), the most accurate model is ERNN 14-25-1 with minimum forecast measure error.

Article information

Search history
Receive :13/12/2020
Date modified :30/12/2020
Publishing :31/12/2020
Available online:5/5/2021

Keyword :
Neural Networks
Forecasting
Stock Prices
Bank of Palestine

Introduction

In fact, the stock market is often considered the primary indicator of a country's economic strength and development. Stock Market is a market where the trading of company stock, both listed securities and unlisted takes place (Patil et al., 2016).

The process of making assumptions of future Changes based on existing data is Forecasting. The more accurate the forecasting, the more it could be helpful to make decisions for future. Forecasting stock price has always been a serious issue in financial fields. Stock market prediction is regarded as a challenging task in financial time-series forecasting because of the fact that stock market indices are essentially dynamic, nonlinear, complicated, nonparametric, and chaotic in nature (Chen et al., 1993). Stock market forecasters focus on developing approaches to successfully forecast predict index values or stock prices, aiming at high profits using well defined trading strategies. The central idea to successful stock market prediction is achieving best results using minimum required input data and the least complex stock market model (George and Kimon, 2009).

Forecasting of future observations based on time series data has received great attention in many fields of research. Several techniques have been developed to address this issue in order to predict the future behavior of a particular phenomenon. The traditional approach based on Box and Jenkins' Autoregressive Integrated Moving Averages (ARIMA) models is commonly used because the resulting models are easy to understand and interpret. When fitting ARIMA models to economic and financial data that are either nonlinear or non-stationary time series, the results of forecasting are expected to be inaccurate. The forecast values tend to converge to the mean of the series after a few forecast values(Okasha, 2014).In recent years, neural networks have been successfully used for modeling financial time series (Zhang and

* Corresponding author : E-mail addresses : shtelbany@gmail.com.
2021 AL – Muthanna University . DOI:10.52113/6/2021-11/8-28.

Michael, 1998). Neural networks are universal function approximators that can map any nonlinear function without a priori assumptions about the properties of the data (Haykin, 1998). Unlike traditional statistical models, neural networks are data-driven, nonparametric weak models, and they let "the data speak for themselves." Consequently, neural networks are less susceptible to the problem of model misspecification as compared to most of the parametric models. Neural networks are also more noise tolerant, having the ability to learn complex systems with incomplete and corrupted data. In addition, they are more flexible and have the capability to learn dynamic systems through a retraining process using new data patterns. So neural networks are more powerful in describing the dynamics of financial time series in comparison to traditional statistical models Kaastra and Milton, 1995).

In the field of prediction for stock market, the most important thing is to improve the prediction accuracy rate (Huang et al, 2006. Chen et al, 2006). Commonly there are many technical analyses for prediction in stock prices. In this paper, we propose an original and universal method by using ANN (MLP, RBFNN, ERNN, JRNN) models with financial analysis for Stock Prices Forecasting to Bank of Palestine.

- **Artificial Neural Networks (ANNs)**

The real appearance of Artificial intelligence was beginning at (1940-1950), (McCulloch and Pitts, 1943) who developed the first computing machines that were intended to simulate the structure of the biological nervous system and perform the logic function through learning. The consequence of this network was a combination of the logic functions, which were used to transfer information from one neuron to another. This eventually led to the development of the binary profit model. According to this model, the neural unit could either switch on or off depending on whether the function was activated or not. A threshold determines the activation of the system; if the input is greater than the threshold, then the neuron is activated and is equal to (1), otherwise, it is (0).

(Hebb, 1949) who developed the first learning rule, which is also known as Hebb learning rule His premise was that if two neurons were active at the same time then the strength between them should be increased. It is based on the simultaneous combination of neurons and is capable of strengthening the connection between them. (Rosenblatt, 1958, 1962) The development of a network based on "Perceptron". It is a single feed-forward network: a unit that produces an output scaled as 1 or -1 depending on the weighted combination of inputs. A major development in ANN occurred when (Cowan, 1967), introduced new functions, such as activation of the smooth sigmoid function, which have the capacity to deal with nonlinear functions more effectively than the learning process "perceptron" model.

(Rumelhart et al., 1986) put forward the back propagation (BP) algorithm of multilayer networks. The procedure which uses the gradient-descent learning technique for multilayer feed-forward ANN is known as back-propagation or the generalized delta rule.

Artificial Neural Networks are one of the areas of Artificial Intelligence that have improved the mechanism of human thinking. ANNs can be considered as mathematical algorithms or a computer model based on biological Neural Networks (brain). The idea of Neural Networks revolves around how to simulate the brain through computers. The main feature of ANN is its ability to learn(Lewis, 2017).

1. Working Principle of An Artificial Neuron

In order to understand the complexity and real operation of artificial neural network, it is necessary to know how the biological nervous system works. The human brain consists of too many processing units, called neurons, and each neuron associated with a large number of other neurons through connections called synapses. Neurons work in parallel and transmit information across them. It is believed that treatment is done by neurons and memory in the synapses, that is, in the way neurons are connected and influence each other (Alpaydin, 2016).

An ANN consists of " neurons" inspired from biological neurons. Artificial neurons are usually organized in layers so that each ANN includes (Pantazis and Alevizakou, 2013):

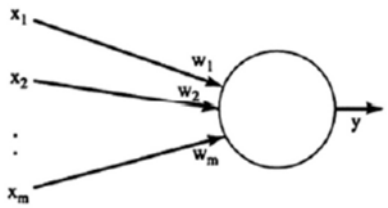
- An input layer: for input data. This layer has as many neurons as the ANN's input variants. Input layer neurons are connected to the neurons of the hidden layers or to neurons in the next layer.
- Hidden layers: Each hidden layer can have n neurons linked in different ways to the other hidden layers or to the output layer. Hidden layer neurons can get their input through the input layer or some other hidden layer or, in some cases, even the output layer.
- An output layer: through which the output vector passes. This layer has as many neurons as the ANN's output variants. Output layer neurons can get their input through the input layer or the hidden layers.

Neuron is a multi-input single output information processing unit and its information processing is non-linear (Zhongzhi, 2012).

- **Artificial Neural Network Terminologies**

The common terminologies used in artificial neural networks (ANN) includes (i) weight, (ii) bias,(iii) threshold, (iv) activation function. etc. In order to understand these terminologies, consider a simple neural network illustrated in Fig.1. As we can see in following Fig.1 . We have inputs $x_1, x_2, \dots, x_m$ coming into the neuron. Each input x_i is multiplied by its corresponding weight w_i then the product $x_i w_i$ is fed into the body of the neuron. The neuron adds up all the products for $i = 1,2, \dots, m$. The weighted sum of the products is usually denoted as a net in the neural network literature.

Figure 1. Simple Neural Network without Bias



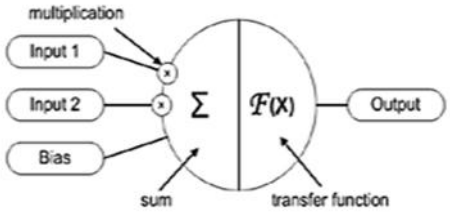
That is, the neuron evaluates the net as the following: $net = x_1 w_1 + x_2 w_2 + \cdots + x_m w_m$ (1)
In general, the net input given by net input = $\sum_i x_i w_i$.

- Weight**:** Weight is basically the information used by the neural network to solve a problem. Neurons connect to each other through some directed communication links, which are associated with weights. The weight may be

used, or it can take random values, or it can be set to zero, or it can be calculated by some methods (Jayaraman et al., 2009).

- **Bias:** Bias is similar to weight. It acts exactly as a weight on a connection from a unit whose activation is always 1. Increasing the bias increases the net input to the unit. The bias improves the performance of the neural network. Similar to the initialisation of the weights, a bias can be initialised to zero or to any specified value based on the neural network. A neural network with bias is illustrated in Fig.2 (Jayaraman et al., 2009).

Figure 2. Simple Neural Network with Bias



The benefit of artificial neuron model simplicity can be seen in its mathematical description Eq. 2:

$$y(k) = F(\sum_{i=0}^{m} w_i\,(k)x_i(k) + b) \qquad (2)$$

where:

x_i is input value in discrete time k where i goes from 0 to m

w_i is weight value in discrete time k where i goes from 0 to m

b is bias

F is a transfer function

$y(k)$ is output value in discrete time k

Finally, the neuron computes its output y as a certain function. This function is called the activation (or sometimes transfer) function.

- Threshold: The threshold θ is a factor which is used in calculating the activations of the given net. Based on the value of threshold, the output may be calculated, i.e., the activation functions is based on the value of θ. For example. the activation functions may be as follows

(Jayaraman et al., 2009):

$$y = f(net) = \begin{cases} -1, & if\ net < \theta; \\ +1, & if\ net \geq \theta; \end{cases} \qquad (3)$$

- Activation Functions **:** Activation functions are the mechanisms by which an artificial neuron processes information and passes it throughout the network. The activation function takes a single number and performs a certain fixed mathematical functional mapping on it, and it is designed to

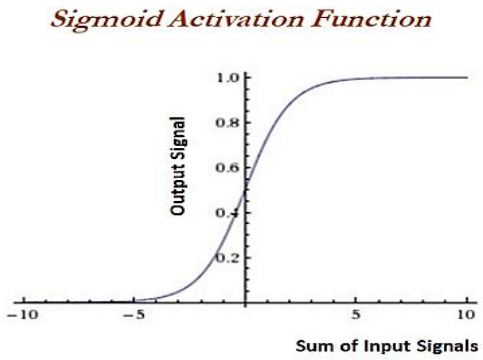
limit the output of the neuron, usually to values between 0 to 1, or -1 to +1. There are many different types of activation functions, the most popular ones are the following (Dangeti, 2017):

- Sigmoid Function**:** Sigmoid has the mathematical form:

$$\sigma(x) = 1/(1 + e^{-X}) \quad (4)$$

It takes a real-valued number and squashes it into a range between 0 and 1, as shown in Fig.3 Sigmoid is a popular choice, which makes calculating derivatives easy and is easy to interpret.

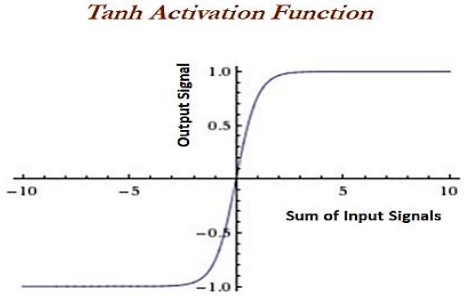
Figure 3. Sigmoid Activation Function



- **Tanh Function:** Tanh squashes the real-valued number into the range [-1, 1]. The output is zero-centered, as shown in Fig.4 . In practice, tanh non-linearity is always preferred to sigmoid non-linearity. Also, it can be proved that tanh is scaled sigmoid neuron

$$\tanh(x) = 2\sigma(2x) - 1 \quad (5)$$

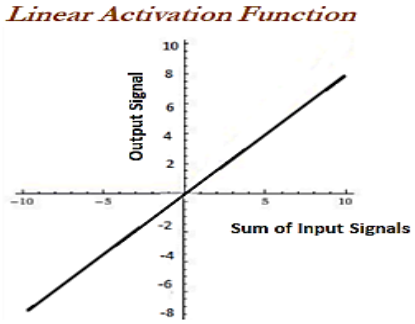
Figure 4. Tanh activation function



- **Linear Function:** The linear activation function multiplies the net input with a constant value k to produce the neuron output, $\varphi(u) = ku \qquad u \in \mathbb{R}$ (6)

The range of the linear activation function is (-∞,∞), as shown in Fig.5 (Moody, 1998).

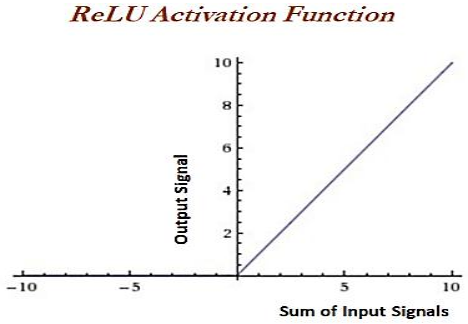
Figure 5. Linear Activation Function



- **Rectified Linear Unit (ReLU) Function:** ReLU has become very popular in the last few years. It computes the function $f(x) = \max(0, x)$. (7)

As shown in Fig.6 . Activation is simply thresholds at zero. Relu is now popularly being used in place of Sigmoid or Tanh due to its better convergence property.

Figure 6. ReLU Activation Function



- **Neural Networks Models**

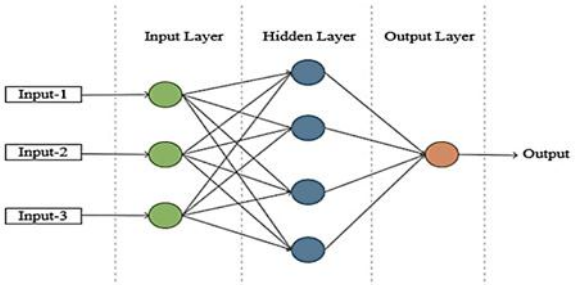
There are many different types of neural network, the most popular ones are the following:

1. Multi-Layered Perceptron (MLP):

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons (with threshold activation). Multilayer perceptrons are sometimes colloquially referred to as "vanilla" neural networks, especially when they have a single hidden layer (Trevor, 2009).

A network is feed-forward when the connections in the network are directed forward, from the input layer towards the output layer and never in the other trend as displayed in Fig.7 or connections within the same layer (Berthold and Hand, 2007). It does not allow any internal feedback of information (Kabundi, 2002). Commonly the input to neurons in a layer is gained from the output of neurons in the promptly previous layer (Kriesel, 2007). If the input layer is straight projected onto the output layer (i.e. there is no hidden layer) the network is said to be a single-layer network, or perceptron. Perceptron is the simplest form of a neural network model. It is composed of an input layer and an output layer (Ohno-Machado, 1996). The following Fig.7 shows a simplified version of a perceptron.

Figure 7. A single-layer perceptron



Among many other learning algorithms, "back-propagation algorithm" is the most popular and the mostly used one for the training of feed-forward neural networks. It is, in essence, a means of updating networks synaptic weights by back propagating a gradient vector in which each element is defined as the derivative of an error measure with respect to a parameter. Error signals are usually defined as the difference of the actual network outputs and the desired outputs. Therefore, a set of desired outputs must be available for training. For that reason, back-propagation is a supervised learning rule. A brief explanation of the back-propagation algorithm to train a feedforward neural network is presented in the following (Sazli,2006).

Let us consider a multilayer feedforward neural network as shown in Figure 8. Let us take a neuron in output layer and call it neuron j. The error signal at the output of the neuron j for nth iteration (Haykin, 1998):

$$e_j(n) = d_j(n) - y_j(n) \tag{8}$$

where d_j is the desired output for neuron j and $y_j(n)$ is the actual output for neuron j calculated by using the current weights of the network at iteration n. For a certain input there is a certain desired output, which the network is expected to produce. Presentation of each training example from the training set is defined as an "iteration". Instantenous value of the error energy for the neuron j :

$$\varepsilon_j(n) = \frac{1}{2}\sum_j e_j^2(n) \tag{9}$$

Since the only visible neurons are the ones in the output layer, error signals for those neurons can be directly calculated. Hence, the instantaneous value, $\varepsilon(n)$, of the total error energy is the sum of all $\varepsilon_j(n)$'s for all neurons in the output layer:

$$\varepsilon(n) = \frac{1}{2}\sum_{j\in Q} e_j^2(n) \tag{10}$$

where Q is the set of all neurons in the output layer.

Suppose there are N patterns (examples) in the training set. The average squared energy for the network :

$$\varepsilon_{av} = \frac{1}{N}\sum_{n=1}^{N}\varepsilon(n) \qquad (11)$$

It is important to note that the instantaneous error energy $\varepsilon(n)$ and therefore the average error energy, ε_{av} , is a function of all the free parameters (i.e., synaptic weights and bias levels) of the network. Back-propagation algorithm, as explained in the following, provides the means to adjust the free parameters of the network to minimize the average error energy, ε_{av} . There are two different modes of back-propagation algorithm: "sequential mode" and "batch mode". In sequential mode, weight updates are performed after the presentation of each training example. One complete presentation of the training set is called an "epoch". In the batch mode, weight updates are performed after the presentation of all training examples, i.e. after an epoch is completed. Sequential mode is also referred as on-line, pattern or stochastic mode. This is the most frequently used mode of operation and explained in the following. Let us start by giving the output expression for the neuron j :

$$y_j(n) = f\left(\sum_{i=0}^{m} w_{ji}(n)\, y_j(n)\right) \qquad (12)$$

where m is the total number of inputs to the neuron j (excluding the bias) from the previous layer and f is the activation function used in the neuron j, which is some nonlinear function. Here w_{j0} equals the bias b_j applied to the neuron j and it corresponds to the fixed input $y_0 = +1$. The weight updates to be applied to the weights of the neuron j is proportional to the partial derivative of the instantaneous error energy $\varepsilon(n)$ with respect to the corresponding weight, i.e. $\partial\varepsilon(n)/\partial w_{ji}(n)$, and using the chain rule of calculus it can be expressed :

$$\frac{\partial\varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial\varepsilon(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial w_{ji}(n)} \qquad (13)$$

From Equations (8), (9) and (11) respectively, Equation (14) is obtained.

$$\frac{\partial\varepsilon(n)}{\partial e_j(n)} = e_j(n) \qquad (14)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \qquad (15)$$

$$\frac{\partial y_j(n)}{\partial w_{ji}(n)} = f'\left(\sum_{i=0}^{m} w_{ji}(n) y_j(n)\right) \frac{\partial\left(\sum_{i=0}^{m} w_{ji}(n) y_j(n)\right)}{\partial w_{ji}(n)}$$

$$\frac{\partial y_j(n)}{\partial w_{ji}(n)} = f'\left(\sum_{i=0}^{m} w_{ji}(n) y_j(n)\right) y_j(n) \qquad (16)$$

Where

$$f\left(\sum_{i=0}^{m} w_{ji}(n) y_j(n)\right) = \frac{\partial f\left(\sum_{i=0}^{m} w_{ji}(n) y_j(n)\right)}{\partial\left(\sum_{i=0}^{m} w_{ji}(n) y_j(n)\right)} \qquad (17)$$

Substituting Equations (13), (14) and (15) in Equation (12) yields Equation (18).

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = e_j(n) f'\left(\sum_{i=0}^{m} w_{ji}(n) y_j(n)\right) y_j(n) \qquad (18)$$

The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the delta rule, given in Equation (19).

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} \qquad (19)$$

In Equation (19), η corresponds to the learning-rate parameter of the back-propagation algorithm, which is usually set to a pre-determined value and kept constant during the operation of the algorithm.

2. Radial Basis Function Neural Networks (RBFNN)

RBF neural networks, a class of feed forward neural networks have universal approximation capabilities. The design of this network is viewed as a curve fitting approximation problem in a high dimensional space. According to this view point, learning is equivalent to finding a surface in a multidimensional space that provides the best fit to the training data. In its most basic form it involves 3 layers with entirely different roles. Input layer is made of source nodes that connect the network to its environment. Second is the hidden layer which applies an on linear transformation from the input space to the hidden space, which is of high dimensionality. Output layer is linear, supplying the response of the network to the activation patterns applied to the input layer (Kumar et al., 2012).

An RBF is symmetrical about a given mean or center in a multidimensional space. Each RBF unit has two parameters, a center x_j , and a width σ_j . This center is used to compare the network input vector to produce a radially symmetrical response. The width controls the smoothness properties of the interpolating function. Response of the hidden layer are scaled by the connection weights of the output layer and then combined to produce the network output. In the classical approach to RBF network implementation, the basic functions are usually chosen as Gaussian

and the number of hidden units is fixed based on some properties of the input data. The weights connecting the hidden and output units are estimated by linear least squares method (LMS) (Haykin, 1998). There are different learning strategies available for the design of an RBF network, depending on how centers of the radial basis functions of the network are specified (Pai et al., 2003) .In the present study, Random initialization method has been used for RBF modeling. The simplest approach is to design RBFNN. In this, the number of radial-basis functions defining the activation functions of the hidden units are fixed. Specifically, the locations of the centers may be chosen randomly from the training data set. The RBFs use Gaussian activation function which is defined as:

$$\phi_j(x) = \exp\left(-\left\|x_j - \xi_i\right\|^2 / 2\sigma_j^2\right) \quad (20)$$

Where x_j is the center and σ_j is the width (standard deviation), j=1, 2,...,c, where c is the number of centers. The only parameter that would need to be learned in this approach is the linear weights in the output layer of the network. The weights are learned using a simple LMS algorithm. The algorithm to train the network by using random initialization method is the following (Pai,2004):

1. Select the number of RBF centers arbitrarily.
2. Initialize their centers from input data randomly.
3. Set $E_{tot} = 0$.
4. Choose the input output pair $\left(\xi_i^{\mu}, \xi_k^{\mu}\right)$, where $\mu = 1,2,3,.....,n$ are the number of patterns and $i = 1,2,3,.....,p$ are the number of input features, k is the output feature.
5. Compute the hidden layer output $\nu_j = e^{\left\|x_j - \xi_i\right\|^2 / 2\sigma_j^2}$,where x_j is the center and σ_j is width of the RBF unit.
6. Compute the output using $O_k = 1/\left(1 + e^{-\sum w_{kj} \nu_j}\right)$.
7. Compute the square error $E = \left(O_K - \xi_K\right) \times \left(O_K - \xi_K\right) and\ E_{tot} = E_{tot} + E$.
8. The change in the output layer weights are calculated as.

$$\partial_k = \left(O_K - \xi_K\right) \times O_K \times \left(O_K - \xi_K\right)$$
$$\Delta w_{kj} = \partial_k \times \nu_j \times \alpha \times \eta \quad (21)$$

Where $\alpha\ and\ \eta$ are learning rate and momentum parameters, and

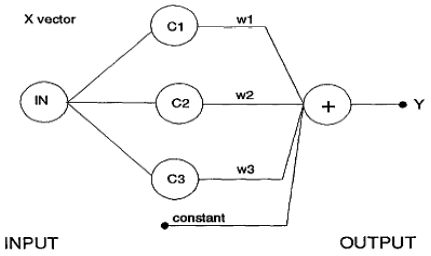
$$w_{kj}^{new} = w_{kj}^{old} + \Delta w_{kj} \quad (22)$$

9. If $E_{tot} \succ E_{\min}$ then go to step 4.

10. Save weights, centers, widths, and exit.

A graphical representation of an RBF neural network with three data centers is shown in Fig.9.

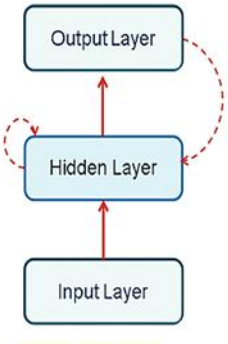
Figure 9. Radial Bias Function Network (Wedding and Cios, 1996)



3. Recurrent Neural Network (RNN)

The recurrent neural network (RNN) is an architecture that has loops in its connections (Mou and Jin, 2018). Recurrent neural networks include the capacity to keep memory, which become Kabundi to learn interim features of the data. This produce frequent networks appropriate to use with data that has a time distance (Kabundi, 2002). In accession to this, feedback loops very much change for the better the learning capacity and rendering of neural networks. Hence, a set of additional context neurons are added to the input layer that receive input from either the output or the hidden layer neurons. These feedback connections to the context neurons have a fixed weight of unity shown in Fig.11 (Kriesel, 2007; Krichene et al., 2016). RNN is very flexible and has been used to solve problems such as speech recognition, language modeling, machine translation, sentiment analysis, and image captioning, time series, to name a few (Gulli and Pal, 2017).

Figure 11. Typical Recurrent Neural Network (Krichene et al., 2016)



An RNN (Williams and Zipser, 1989 ; Rodriguez et al., 1999), is a class of artificial neural network that extends the conventional feedforward neural network with loops in connections. Unlike a feedforward neural network, an RNN is able to process the sequential inputs by having are current hidden state whose activation at each step depends on that of the previous step. In this manner, the network can exhibit dynamic temporal behavior.

Given a sequence data $x=(x_1,x_2,...,x_T)$,where x_i is the data at *i*th time step, an RNN updates its recurrent hidden state h_t by

$$h_t = \begin{cases} 0, & if\ t = 0 \\ \varphi(h_{t-1}, x_t), & otherwise \end{cases} \quad (23)$$

Where φ is a nonlinear function, such as a logistic sigmoid function or hyperbolic tangent function. Optionally, the RNN may have an output $y=(y_1,y_2,...,y_T)$. For some tasks, such as hyperspectral image classification, we need only one output, i.e., y_T.

In the traditional RNN model, the update rule of the recurrent hidden state in (23) is usually implemented as follows:

$$h_t = \varphi(w_{x_t} + Uh_{t-1}) \quad (24)$$

Where W and U are the coefficient matrices for the input atthe present step and for the activation of recurrent hidden units at the previous step, respectively. In fact, an RNN can model a probability distribution over the next element of the sequence data, given its present state h_t, by capturing a distribution over sequence data of variable length. Let $p(x_1,x_2,...,x_T)$be the sequence probability, which can be decomposed into:

$$p(x_1, x_2, ..., x_T) = p(x_1).....p(x_T \mid x_1, x_2, ..., x_{T-1}) \quad (25)$$

Then, each conditional probability distribution can be modeled with a recurrent network:

$$p(x_T \mid x_1, x_2, ..., x_{T-1}) = \varphi(h_t) \quad (26)$$

Where h_t is obtained from (23) and (26).

Training through RNN:

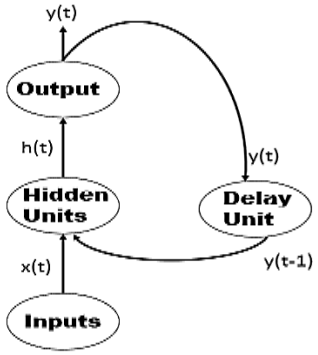
1. A single time step of the input is provided to the network.
2. Then calculate its current state using set of current input and the previous state.
3. The current h_t becomes h_{t-1} for the next time step.
4. One can go as many time steps according to the problem and join the information from all the previous states.
5. Once all the time steps are completed the final current state is used to calculate the output.
6. The output is then compared to the actual output i.e the target output and the error is generated.
7. The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

There are different types of (RNN) architectures such as Jordan Recurrent Neural Network (JRNN), and Elman Recurrent Neural Network (ERNN). In the following, we will discuss the difference among those architectures:

- Jordan Recurrent Neural Network (JRNN)

One of the most famous recurrent neural networks is Jordan RNN (JRNN), where three layers are used, with the addition of a context-layer which receives input from the output-layer, see Fig.12 . This added layer works as a dynamic memory as it saves the previous stales of the output-layer (Krichene et al.,2016). This kind of characteristic results in better model and thus a better and more accurate forecasting (Putri et al., 2018).

Figure 12. Jordan Recurrent Neural Network (Lewis, 2017)

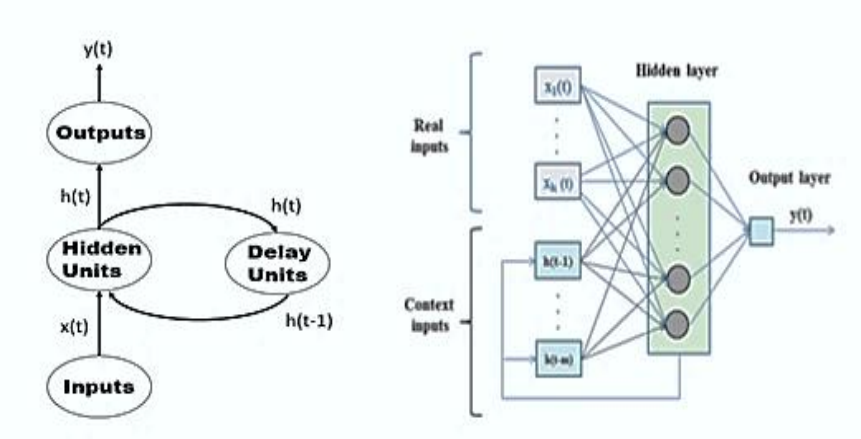


where: $x(t)$ and $y(t)$ are the input and output of the network, respectively, at a discrete time t. $h(t-1)$ and $h(t)$ are the nodes of the context and the hidden layers, respectively (Lewis, 2017).

- Elman Recurrent Neural Network (ERNN)

Another type of RNN is the Elman RNN, It is a popular simple recurrent neural network that has delivered outstanding performance in a wide range of applications. Elman recurrent neural networks are composed of an input layer, a context layer (also called a recurrent or delay layer see Fig.13, a hidden layer, and an output layer. Each layer contains one or more neurons which propagate information from one layer to another by computing a nonlinear function of their weighted sum of inputs. In an Elman neural network, the number of neurons in the context layer is equal to the number of neurons in the hidden layer. In addition, the context layer neurons are fully connected to all the neurons in the hidden layer (Lewis, 2017).

Figure 13. Elman Neural Network (Lewis, 2017 ; Krichene et al., 2016)



where : $x(t)$ and $y(t)$ are the input and output of the network, respectively, at a discrete time t. $h(t-1)$ and $h(t)$ are the nodes of the context and the hidden layers, respectively. Compare Fig.12 (Jordan) with Fig.13 (Elman). They are very similar. However, in a Jordan network the context layer (delay unit) is directly connected to the input of the hidden layer. Like the Elman neural network, the Jordan neural network is useful for predicting time series observations which have a short memory (Lewis, 2017).

1. Performance Tests:

In order to compare forecasting performance of different models, many statistical measures can be used for this purpose, Smaller values indicate better model performance.:

• Mean absolute percentage error (MAPE):

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y_t - \bar{y}_t}{y_t}\right| * 100 \qquad (27)$$

• Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - \bar{y}_t| \qquad (28)$$

• Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{(y_t - \bar{y}_t)^2}{n}} \qquad (29)$$

2. Results:

The available time series data of monthly stock prices for bank of Palestine, from Nov. 2005 to Oct. 2020. The data had 180 observations. The time series data were divided into two parts. That is, 90% of

the series was considered a training set in the period (Nov. 2005 – May. 2019), and 10% was used as a test set in the period (Jun. 2019 – Oct. 2020). which was used to Evaluation of the Performance of Neural Networks Models (MLP, RBFNN, ERNN, JRNN). We used R Package to analyze the data.

All the Neural Networks Models Have:

- learning rate = 0.1
- Iteration = 100
- Three layers:

1. input layer which consist of 14 node.
2. Hidden layer which consist of many neurons (5, 8, 10, 15, 20, 25), except for RBFNN model (No results more than 10 hidden layer)
3. output layer which is only one output.

- Training period: (Nov. 2005 – May. 2019).
- Forecasting period: (Jun. 2019 – Oct. 2020).
- Estimate the MAPE, MAE and RMSE for each model Based on actual (test set) and forecast (forecasting period).

Table 1. Performance tests for MLP

Model	MAPE	MAE	RMSE
MLP 14-5-1	6.307233	0.1172743	0.1400422
MLP 14-8-1	6.799320	0.1270651	0.1514591
MLP 14-10-1	6.509877	0.1212526	0.1447930
MLP 14-15-1	5.560989	0.1027352	0.1256453
MLP 14-20-1	5.489736	0.1013145	0.1234494
MLP 14-25-1	4.619789	0.0848754	0.1107915

Table 2. Performance tests for RBFNN*

Model	MAPE	MAE	RMSE
RBFNN 14-5-1	2.210855	0.03915445	0.06561904
RBFNN 14-8-1	1.924838	0.03408945	0.05941890
RBFNN 14-10-1	2.123007	0.03808812	0.08360779

*Note: No results more than 10 hidden layer.

Table 3. Performance tests for ERNN

Model	MAPE	MAE	RMSE
ERNN 14-5-1	2.477215	0.04374118	0.06440754
ERNN 14-8-1	3.075220	0.05645000	0.10205930
ERNN 14-10-1	4.025349	0.07514163	0.12845460
ERNN 14-15-1	3.201759	0.05878273	0.09710963
ERNN 14-20-1	5.644269	0.10102410	0.13845330
ERNN 14-25-1	1.425869	0.02540497	0.03293498

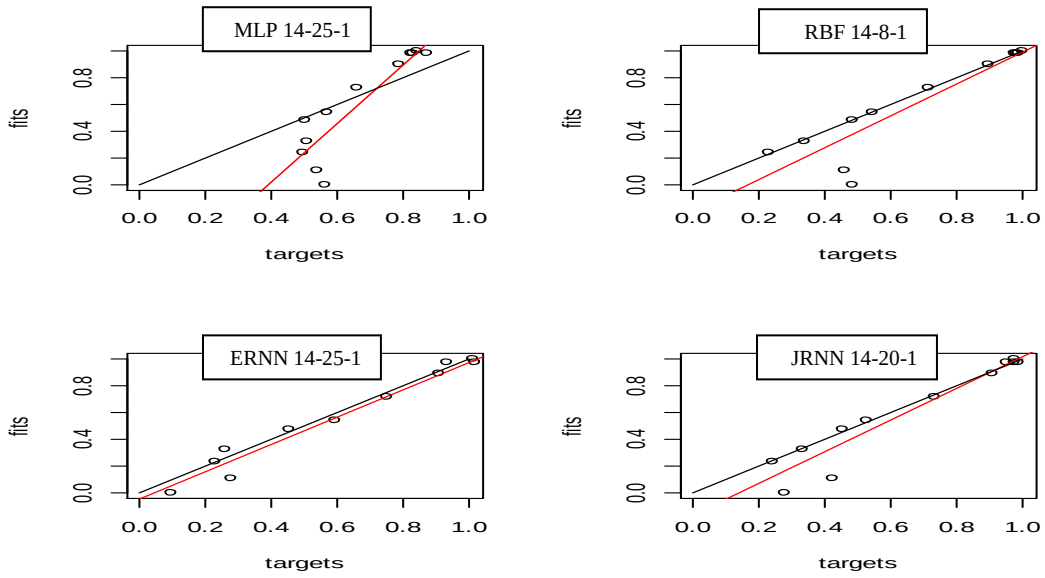
Table 4. Performance tests for JRNN

Model	MAPE	MAE	RMSE
JRNN 14-5-1	2.518542	0.04248686	0.06721032
JRNN 14-8-1	2.776029	0.04635850	0.07662006
JRNN 14-10-1	2.156223	0.03621474	0.0584987
JRNN 14-15-1	2.492915	0.04079090	0.08315895
JRNN 14-20-1	1.733546	0.02829265	0.05594429
JRNN 14-25-1	2.183911	0.03621168	0.06153536

Table 5. Comparison the performance of Artificial Neural models

Model	MAPE	MAE	RMSE
MLP 14-25-1	4.619789	0.08487540	0.11079150
RBFNN 14-8-1	1.924838	0.03408945	0.05941890
ERNN 14-25-1	1.425869	0.02540497	0.03293498
JRNN 14-20-1	1.733546	0.02829265	0.05594429

Figure 14. Regression error plots (Fits Versus Targets)



MAPE, MAE, and RMSE accuracy measures were used to compare the Performance of the different models. The smaller the error values, the better the performance and the predicted values are closer to the actual values. It can be concluded from the above that results of ERNN 14-25-1 model were more accurate with the lowest (MAPE, MAE, and RMSE) and is the most efficient forecasting technique for monthly Stock Prices Forecasting to Bank of Palestine .

3. Conclusion:

The results of applying the (MLP, RBFNN, ERNN, JRNN) Models were compared through the MAPE, MAE and RMSE, the most accurate model is ERNN 14-25-1 with minimum forecast measure error for monthly Stock Prices Forecasting to Bank of Palestine . In future studies, I advise to compare between ANN's and other methods of forecasting, such as the SVM and GARCH models to determine which one is more efficient in the volatility data case (economic and financial data), and I advise to use a hybrid method of ANN's and other methods.

References:

Alpaydin, E. (2016). Machine learning: the new AI. The MIT Press.

Berthold, M. R., and Hand, D. J. (2007). Intelligent data analysis: An introduction. Springer. 2nd Edition.

Chen, S., Mulgrew, B., Grant, P. M. (1993). A clustering technique for digital communications channel equalization using radial basis function networks. IEEE Transactions on Neural Networks, 4(4): 570-578 .

Chen, W. H., and Shih, J. Y., (2006). A study of Taiwan's issuer credit rating systems using support vector machines. Expert Systems with Applications, 30: 427-435.

Cowan, J. D. (1967). A Mathematical Theory of Central Nervous Activity; unpublished Ph.D. dissertation; University of London.

Dangeti, P. (2017). Statistics for Machine Learning: Techniques for exploring supervised, unsupervised, and reinforcement learning models with Python and R. Packt Publishing.

George S. A., and Kimon, P. V. (2009). Forecasting stock market shortterm trends using a neuro-fuzzy based methodology, Expert Systems with Applications, 36(7): 10696–10707 .

Gulli, A., & Pal, S. (2017). Deep Learning with Keras. Packt Publishing Ltd.

Haykin, S. (1994). Neural networks: a comprehensive foundation. Prentice Hall PTR. 1st Edition.

Haykin, S. (1998). Neural Networks: A Comprehensive foundation. 2nd edition. Prentice Hall.

Hebbs, D. G. (1949). The organization of behavior. Wiely and Sons, New York, NY, USA.

Huang, C. L., and Tsai, C. Y., (2009). A hybrid SOFM-SVR with a filter-based feature selection for stock market forecasting, Expert Systems with Applications: An International Journal, 36(2):1529-1539.

Jayaraman, S., Esakkirajan, S.,& Veerakumar, T.(2009). Digital Image Processing. Tata McGraw-Hill Education.

Kaastra, I. and Milton, S. B. (1995). Forecasting futures trading volume using neural networks, Journal Futures Markets, 15(8): 853–970 .

Kabundi, A. (2002). Macroeconomic Forecasting A Comparison Between Artificial Neural Networks and Econometric Models, MA. thesis, Rand African University.

Krichene, E., Masmoudi, Y., Alimi, A. M., Abraham, A., & Chabchoub, H. (2016, December). Forecasting Using Elman Recurrent Neural Network. In International Conference on Intelligent Systems Design and Applications (pp. 488-497). Springer, Cham.

Kriesel, D. (2007). A Brief Introduction to Neural Networks. [Online] Available at: http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf

Kumar, S., Pai, P. S., and Rao, B. R. (2012). Radial Basis Function Network Based Prediction of Performance and Emission Characteristics in a Bio Diesel Engine Run on WCO Ester. Hindawi Publishing Corporation, dvances in Artificial Intelligence : 1–7.

Lewis, N. D. (2017). Neural networks for time series forecasting with R: An intuitive step by step blueprint for beginners. CreateSpace Independent Publishing Platform.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

Moody, J. (1998). Forecasting the economy with neural nets: A survey of challenges and solutions. In Neural Networks: tricks of the trade (pp.347-371). Springer, Berlin, Heidelberg.

Mou, L., & Jin, Z. (2018). Tree-Based Convolutional Neural Networks Principles and Applications. Springer.

Ohno-Machado, L. (1996). Medical applications of artificial neural networks: connectionist models of survival .Doctoral dissertation, Stanford University.

Okasha, M. K. (2014). Using Support Vector Machines in Financial Time Series Forecasting. International Journal of Statistics and Applications, 4(1): 28-39 .

Pai, P. S. (2004). Acoustic emission based tool wear monitoring using some improved neural network methodologies, [Ph.D. thesis], S.J. College of Engineering, University of Mysore, Mysore, India.

Pai, P. S., Nagabhushan, T. N. and Rao, B. R. (2003). Radial basis function neural networks for tool wear monitoring. International Journal of COMADEM, 5(3): 21–30.

Pantazis, G., & Alevizakou, E. G. (2013). The use of artificial neural networks in predicting vertical displacements of structures. International Journal of Applied Science and Technology, 3(5).

Patil, S. S., Patidar, K., Jain, M., (2016). Stock Market Prediction Using Support Vector Machine. International Journal of Current Trends in Engineering & Technology, 2(1): 18-25

Putri, T. E., Firdaus, A. A., & Sabilla, W. I. (2018). Short-Term Forecasting of Electricity Consumption Revenue on Java-Bali Electricity System using Jordan Recurrent Neural Network. Journal of Information Systems Engineering and Business Intelligence, 4(2), 96-105.

Rodriguez, P., Wiles, J. and Elman, J. L. (1999). A recurrent neural network that learns to count. Connection Sci., 11(1): 5–40.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386-408

Rosenblatt, F. (1962). Principles of Neurodymanics: Perceptrons and the Theory of Brain Mechanisms. Spartan books.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagate on (Vol. 1). Cambridge: MA. MIT . Press; 1: 318-362.

Sazli, M. H. (2006). A brief review of feed-forward neural networks. Communications Faculty Of Science University of Ankara,50(1):11-17.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks, Neural Comput. 1(2): 270–280.

Zhang, G. Q., and Michael, Y. H. (1998). Neural network forecasting of the British Pound/U.S. Dollar Exchange Rate, Omega, 26(4): 495–506 .

Zhongzhi, S. (2012). Intelligence science. (Vol. 2). World Scientific.

Appendix: Monthly Stock Prices to Bank of Palestine.

11/2005	1.610	11/2008	1.244	11/2011	1.761	11/2014	2.298	11/2017	2.402
12/2005	1.600	12/2008	1.201	12/2011	1.798	12/2014	2.315	12/2017	2.598
01/2006	1.500	01/2009	1.351	01/2012	1.786	01/2015	2.315	01/2018	2.549
02/2006	1.430	02/2009	1.283	02/2012	1.860	02/2015	2.298	02/2018	2.569
03/2006	1.540	03/2009	1.556	03/2012	1.904	03/2015	2.282	03/2018	2.618
04/2006	0.850	04/2009	1.550	04/2012	1.973	04/2015	2.396	04/2018	2.471
05/2006	0.820	05/2009	1.473	05/2012	1.849	05/2015	2.270	05/2018	2.451
06/2006	0.720	06/2009	1.710	06/2012	1.869	06/2015	2.261	06/2018	2.451
07/2006	0.710	07/2009	1.741	07/2012	1.807	07/2015	2.333	07/2018	2.431
08/2006	0.810	08/2009	1.901	08/2012	1.786	08/2015	2.261	08/2018	2.412
09/2006	0.720	09/2009	1.886	09/2012	1.800	09/2015	2.261	09/2018	2.333
10/2006	0.750	10/2009	1.912	10/2012	1.953	10/2015	2.333	10/2018	2.275
11/2006	1.020	11/2009	1.896	11/2012	1.939	11/2015	2.487	11/2018	2.265
12/2006	1.220	12/2009	1.938	12/2012	2.070	12/2015	2.713	12/2018	2.275
01/2007	1.490	01/2010	1.891	01/2013	2.077	01/2016	2.650	01/2019	2.157
02/2007	1.650	02/2010	1.912	02/2013	2.174	02/2016	2.667	02/2019	2.255
03/2007	1.538	03/2010	1.901	03/2013	2.202	03/2016	2.495	03/2019	2.230
04/2007	1.144	04/2010	1.912	04/2013	2.232	04/2016	2.428	04/2019	2.170
05/2007	1.426	05/2010	1.808	05/2013	2.170	05/2016	2.380	05/2019	2.160
06/2007	1.356	06/2010	1.783	06/2013	2.163	06/2016	2.390	06/2019	2.000
07/2007	1.426	07/2010	1.690	07/2013	2.163	07/2016	2.409	07/2019	2.120
08/2007	1.296	08/2010	1.669	08/2013	2.193	08/2016	2.409	08/2019	2.080
09/2007	1.197	09/2010	1.886	09/2013	2.232	09/2016	2.457	09/2019	2.040
10/2007	0.952	10/2010	1.746	10/2013	2.263	10/2016	2.390	10/2019	2.020
11/2007	1.098	11/2010	1.741	11/2013	2.372	11/2016	2.447	11/2019	2.000
12/2007	1.028	12/2010	1.757	12/2013	2.480	12/2016	2.533	12/2019	2.000
01/2008	1.058	01/2011	1.783	01/2014	2.465	01/2017	2.581	01/2020	2.000

02/2008	1.237	02/2011	1.736	02/2014	2.519	02/2017	2.629	02/2020	2.010
03/2008	1.383	03/2011	1.772	03/2014	2.449	03/2017	2.581	03/2020	1.961
04/2008	1.502	04/2011	1.907	04/2014	2.398	04/2017	2.402	04/2020	1.881
05/2008	1.665	05/2011	1.891	05/2014	2.398	05/2017	2.402	05/2020	1.800
06/2008	1.665	06/2011	1.873	06/2014	2.315	06/2017	2.451	06/2020	1.770
07/2008	1.716	07/2011	1.879	07/2014	2.307	07/2017	2.422	07/2020	1.700
08/2008	1.720	08/2011	1.860	08/2014	2.315	08/2017	2.422	08/2020	1.660
09/2008	1.630	09/2011	1.860	09/2014	2.340	09/2017	2.441	09/2020	1.600
10/2008	1.442	10/2011	1.848	10/2014	2.315	10/2017	2.441	10/2020	1.550

Source: Palestine Exchange: https://web.pex.ps/