*Malaysian Journal of Applied Sciences* 2018, **Vol 3(2)**: 8-20
© Universiti Sultan Zainal Abidin
eISSN 0127-9246 (Online)

**Malaysian Journal of Applied Sciences**

**ORIGINAL ARTICLE**

# Computing the Performance of FFNN for Classifying Purposes

***Hadaate Ullah** [a,b]**, Adnan Kiber** [b]**, Asadul Huq** [b] **and Mohammad Arif Sobhan Bhuiyan** [c]*

[a]Electrical and Electronic Engineering, University of Dhaka, Bangladesh.
[b]Electrical and Electronic Engineering, Southern University Bangladesh, Bangladesh.
[c]Electrical and Electronics Engineering, Xiamen University Malaysia, Sepang, Selangor, Malaysia.

*Corresponding author: hadaate@southern.edu.bd / sendbablu.apee@gmail.com

*Received: 04/02/2018, Accepted: 01/11/2018*

## Abstract

Classification is one of the most hourly encountered problems in real world. Neural networks have emerged as one of the tools that can handle the classification problem. Feed-Forward Neural Networks (FFNN's) have been widely applied in many different fields as a classification tool. Designing an efficient FFNN structure with the optimum number of hidden layers and minimum number of layer's neurons for a given specific application or dataset, is an open research problem and more challenging depend on the input data. The random selections of hidden layers and neurons may cause the problem of either under fitting or over fitting. Over fitting arises because the network matches the data so closely as to lose its generalization ability over the test data. In this research, the classification performance using the Mean Square Error (MSE) of Feed-Forward Neural Network (FFNN) with back-propagation algorithm with respect to the different number of hidden layers and hidden neurons is computed and analyzed to find out the optimum number of hidden layers and minimum number of layer's neurons to help the existing classification concepts by MATLAB version 13a. By this process, firstly the random data has been generated using an suitable matlab function to prepare the training data as the input and target vectors as the testing data for the classification purposes of FFNN. The generated input data is passed on to the output layer through the hidden layers which process these data. From this analysis, it is find out from the mean square error comparison graphs and regression plots that for getting the best performance form this network, it is better to use the high number of hidden layers and more neurons in the hidden layers in the network during designing its classifier but so more neurons in the hidden layers and the high number of hidden layers in the network makes it complex and takes more time to execute. So as the result it is suggested that three hidden layers and 26 hidden neurons in each hidden layers are better for designing the classifier of this network for this type of input data features.

**Keywords:** ANNs, Back-Propagation algorithm, Tan Sigmoid Activation Function, Regression plot, classifier.

## Introduction

The neurons of Artificial Neural Networks (ANNs) are like the neurons of biological model (where there is no specific relationship between the inputs and outputs and it is not easy to formulate the mathematical model). Neural networks (NNs) are very well known for finding out the regression and classification problems in huge fields (Hagan et al.,1995) because of its no require any elaborated information regarding the system which conducts like a black box (Danaher et al., 2004). ANNs, shown in Figure 1, compose of input, hidden and output layers (Engelbrecht, 2007). ANN generally gets information from other ANNs or neighboring and gathers this information by transfer function using sum which control the firing and strength of the exciting signals as a function of their respective weights and biases.

If a NN consists only by a single hidden layer is called Perceptron that is the simplest but not able to find out nonlinearly separable problems. On contrary, Multilayer Perceptron (MLP) is more useful to solve the nonlinearly separable problems. The merits of neural network come from its ability for recognizing and nonlinear relationships of model among data. ANNs are easiest way for dealing nonlinear relationships and clustering of real data compare to strict linear relationship (Naguib and Sherbet, 2001). But ANNs designers face two problems, one is the network structure and another is the network generalization. ANNs design needs application-specific appropriate architecture that includes network type, number of layers, number of neurons (nodes) in hidden layers, and activation functions (transfer functions) between layers.

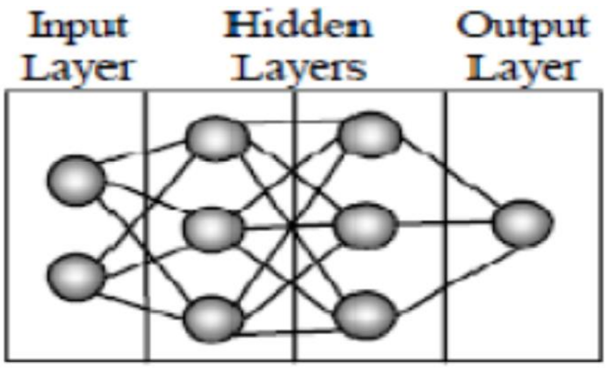


**Figure 1.** Neural network architecture

The transfer potential of an ANN, $A_i$ is equal to

$$A_i = \sum_{i=1}^{N} w_{ij} x_j - b_i \tag{1}$$

Where $N$ is the amount of elements of input vector, $\omega_{ij}$ are the interconnection weights, and $b_i$ is the bias of the neurons (Haykin, 1994).), the bias is the coefficient that controls the transfer of the signal followed by the ANNs. The output of neurons trusts only on the information which is natively available in the neurons that is either stored in the inside or arrived through the weighted of coefficients. The output of neurons, $y_i$ is computed by the summing of the weighted inputs along with a bias through a transfer function in below:

$$y_i = \varphi(Ai) = \sum_{i=1}^{N} w_{ij} x_j - b_i \tag{2}$$

The transfer function is desired to confine the neuron output within the rage [0, 1] or [–1, +1] (Pacelli and Azzollini, 2011). The most popular transfer functions are the linear combination, sigmoid function, and step function. In the linear combination, a number of linear neurons execute a linear transformation by the input vector, $y_i = \Phi(A_i) = kA_i$, where $k$ is a scale factor. A sigmoid transfer function that is continuous and differentiable generates an output within the limit [0,1]. For this reasons, it is used in ANN models where the learning algorithm demands derivatives. Usually, sigmoid transfer function represents to the particular issue through the logistic function followed by the following formula:

$$\varphi(A_i) = \frac{1}{1+e^{-KA}} \tag{3}$$

Where $k$ is the constant by which the shape of a curve is controlled. The sigmoid is that logistic function which has ability to evaluate the derivative easily and it is more important when evaluating the updates of weights in the network. Thus it makes the network easier manipulable in mathematically that is more attractive to early scientists for minimizing the computing load of their works.

Before using the neural network to solve the specific problem, its weights and biases are to be adjusted. This function is performed by a learning algorithm that repetitively updates the weights until a predefined value is achieved and the learning algorithm is stopped. There are mainly two learning algorithm (Angelini et al., 2008), (i) supervised learning, and (ii) unsupervised learning algorithm. Supervised learning algorithm is distinguished by a training set consist of inputs and the respective desired outputs where the produced error is applied for updating the weights in backward direction. In unsupervised learning algorithms, the network is composed only by a set of inputs and without their corresponding desired output that guides the network to self-organize and adjusting its weights. This learning algorithm is mainly used in data mining and clustering where big data analysis is main concern.

Many studies had been carried out on artificial neural networks for classifying the biomedical images. Based on studies, ANNs are categorized as FFNN, Cascade-Forward Neural Network, Nonlinear Auto-Regression Neural Network, Generalized Regression Neural Network (GRNN), Recurrent Neural Network (RNN), Radial Basis Function Neural Network (RBFNN) and Probabilistic Neural Network (PNN). All these are application specific. For biomedical image classification, FFNN with back-propagation learning algorithm, also known as a Feed-Forward Back propagation neural network (FFBPNN) is usually preferred because of its high accuracy and less iterations period (Rani and Vashisth, 2016). It is very simple and effective model of ANNs where a lot of input and target pairs are required for training and testing phases (Amardeep and Swamy, 2017). As it works only in multilayer, so it is also called multilayer FF neural network (Rodan et al., 2016).

A feed forward neural network (FFNN) is a network where each neuron of a layer is associated to all the neurons of the next layer. This topology opposes backward connections that are set in numerous recurrent neural networks (Han et al., 2012; Rodan and Tiˇno, 2011; Jaeger, 2007; Jaeger, 2002; Lin et al.,1996;), layer-skipping, and in application specific neural network architectures like Fully Connected Cascade (FCC) (Wilamowski, 2009). Another important issue of this network is that no delay is permitted which make the network more meaningful only to illustrate the static models. Various studies exhibited that in spite of lacking dynamic abilities, a FFNN is used to illustrate the function mapping and its derivatives (Hornik et al.,1990). Nonetheless, the choosing suitable number of hidden layers and neurons in the layers of FFNN is more challenging in a general significance computing backgrounds (e.g. Matlab) and embedded environments (e.g. microcontrollers and Field Programmable Gate Array(FPGA)) as a function of its training proficiency and estimation cost during training.

There are different types of training algorithm in NN such as Gradient Descent (function: traingd), Bayesian Regularization (function: trainbr), Variable Learning Rate Gradient Descent (function: traingdx), Gradient Descent with Momentum (function: traingdm), and Train

Levenberg Marquartdt Algorithm (LMA) (function: trainlm). When the network weights and biases are initialized, the network is prepared for training that demands a set of inputs and target outputs. During the training, the weights and biases of the network are automatically adjusted to minimize the network performance. The performance functions are Mean absolute error performance function (mae), Mean squared error performance function (mse), Mean squared error w/reg performance function(msereg), and Sum squared error performance function (sse). The default performance function for FFNN is mean squared error (mse). There are also various learning functions in NN such as Conscience bias learning function (learncon), Gradient descent weight/bias learning function (learngd), Gradient descent momentum learning function (learngdm), and Perceptron weight/bias learning function (learnp). No single training algorithm is applicable in all types of solution because these algorithm is application specific. Similar concept is applicable for learning algorithm and performance functions. Corte-Valiente et al. (2017) evaluates the performance of these types of training algorithms in a FFNN for analyzing the whole uniformity in the lighting systems of outdoor and they found the minimum error with LMA (Valiente et al., 2017).

Though FFNNS are used in a variety of purposes with great a success but the main demerits of this kind of networks is that there is no surety that this model will perform good for the all of kind problems at hand (Benardos and Vosniakos, 2007). Therefore, the development of FFNNs depends on the search of the best combination of the following four elements that consists its architecture (Cybenko,1989):

1. The number of layers
2. The number of neurons in each layer
3. The activation function
4. The training algorithm

A multilayer perceptron network (MPN) consists of an input layer, one or more hidden layers of computation nodes, and an output layer. Figure 1 shows a typical feed-forward network with two hidden layers consisting of three nodes in each hidden layers, two input neurons and one output. As mentioned in (Ahmed and Noureldien, 2014), determining number of hidden layers and number of neurons in each layer of FFNN is a challenge depends on its specific-application. The random selections of hidden neurons and hidden layers may cause the problem of either under fitting or over fitting of data.

So, during any application of neural networks for the classification, the same question always rises; how many hidden layers and how many nodes in each layer should be used? Although it has been almost two decades now since the first introduction of neural networks in remote sensing (Benediktsson et al., 1990) there exists no exact method to answer this question (Mas and Flores, 2008), and it is a critical question since the selection of topology has a profound impact on classification results. Traditionally identification of topology has been based on trial and error. Hecht-Nielsen (Nielsen,1987) imported that any continuous function can be represented by a neural network that has only one hidden layer with exactly 2n + 1 nodes, where n is the number of input nodes. This is not the case however as Hecht-Nielsen stated that the 2n + 1 rule is not for any class of activation functions but for a specific one. This activation function is much more complex, compared with the commonly used sigmoidal functions. It has been suggested (Kurkova, 1992) that two hidden layers should be used to compensate for lost efficiency when using regular activation functions. The argument that it is sufficient to use a single hidden layer still holds when using regular transfer functions (e.g. sigmoidal) but the number of required hidden nodes can be as high as the number of training samples (Huang, 2003; Huang and Babri,1997). The purpose of using a second hidden layer is to drastically reduce the total required number of hidden nodes. Huang (2003) proved that in the two-hidden-layer case, with m output neurons, the number of hidden nodes that are enough to learn N samples with negligibly small error is given by:

$$2\sqrt{(m+2)N} \tag{4}$$

Specifically, he suggests that the sufficient number of hidden nodes in the first layer is

$$2\sqrt{N/(m+2)}+\sqrt{(m+2)N} \tag{5}$$

and in the second it is

$$m\sqrt{N/(m+2)} \tag{6}$$

There are a lot of methods to find and the number of hidden layers and the numbers of hidden nodes in a layer, some of them are explained in (Panchal and Panchal, 2015; Stathakis, 2009). Though most of the research has been in the field of FFNN and it is believed that FFNNs are static input-output mapping representing a non-linear system but the hidden nodes in RNN are assumed to be constant (Lin et al., 2013). The challenge here in FFNN is to identify the optimal combination of hidden layers and neurons in each layer which will generate minimum error in lesser duration of iteration.

So, for designing the FFNNs, one crucial and difficult challenge is to determine the number of hidden layers and the number of neurons in the hidden layers. The hidden layer is responsible for internal representation of the data and the information transformation from input to output layer. If there are too few neurons in the hidden layer, the network may not contain sufficient degrees of freedom to form a representation. Again if too many neurons are defined, the network might become over trained. Therefore, an optimum design for the number of neurons in the hidden layer is required. In this research, it is tried to analyze the effect and how many number of hidden layers and the number of neurons in hidden layers are required and what combination of its of FFNN to make its suitable application in classification purposes by counting its performance.

## Methodology

FFNN with back propagation algorithm (FFBPNN) has the better accuracy and precision compared to than other techniques in classifying application. FFBP NN computes output in forward and error in backward direction. In forward processing, input layer is used for taking the input data that is passed on to the hidden layers, and hidden layer processes the data. Output is taken from the output layer which is shown in Figure 2.

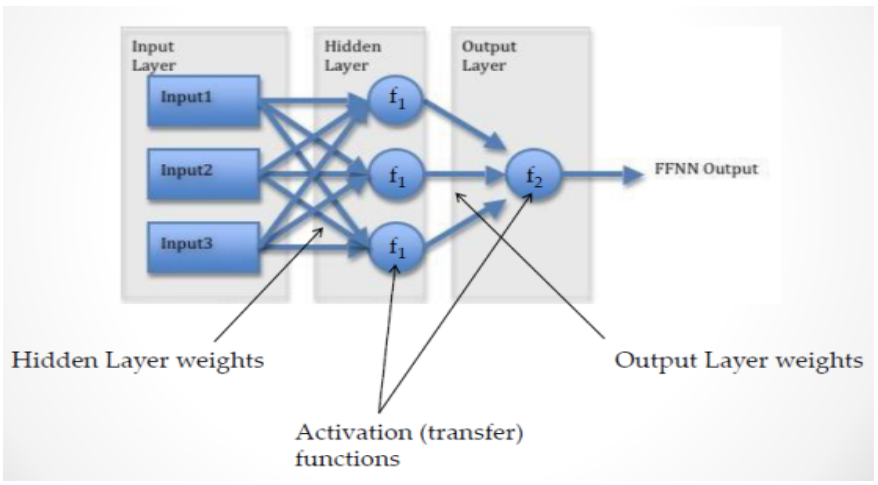


**Figure 2.** FFNN General Architecture

FFNN with back propagation learning is the most commonly used technique for training the ANN to minimize the gradient. For this research, Neural Network Toolbox™ of MATLAB (Version 2013a) has been used. The design algorithm of this network for the classification purposes is as follows:

Step 1: Generate the data randomly.
Step 2: Create the network.
Step 3: Configure the network.
Step 4: Initialize the weights and biases randomly.
Step 5: Train the network and feed the training sample.
Step 6: Propagate the inputs to the forward.
Step 7: Back propagate the error to the hidden layer.
Step 8: Validate the network.
Step 9: Use and analysis the performance of the network.

After generating the data for preparing the training data which is introduced to this network as input and target vectors as the testing data for the classification purposes the network for FFNN is created by an appropriate MATLAB function with the different number of hidden layers and the different number of neurons in each hidden layers where the weights and biases are initialized randomly and also it demands three arguments to return the network objectives. The first is a matrix of input vectors and the second is a matrix of target vectors where these two are required for setting up the network dimensions and parameters. The last is an array bearing the sizes of each hidden layer (where the output layer size is equal to the number of target categories). Two input vectors means the two features or parameters of images and four target means four classes in this algorithm. After creating the network, the training processing is started by a suitable function of MATLAB that is used. Then the transfer function (Tan Sigmoid) is identified for training the data which is used in each layer. Transfer functions evaluate the layer's output from its input by using this training Algorithm. During simulation, the trial and error concept is followed. At first the initial number of hidden layers, neurons in the hidden layers and activation function find out, and the error of performance are recorded. After that for the same architecture, first the numbers of neurons in the hidden layers are changed and the error of performance using MSE and both are also recorded. And then the number of hidden layers is changed and the error of performance and both are also recorded. This procedure is repeated, and the number of hidden layers and neurons in the hidden are selected which provides the least error of performance. Based on this algorithm, the architecture of this network including input, output and hidden layers is shown in **Figure 3.**

From this figure, it is clear that input and output layer has two nodes (input features) and four nodes (output classes) respectively. There are three successive hidden layers which contains 36 neurons, where learning rule is back propagation and training algorithm is Levenberg Marquartdt. All the neurons are made to pass from the input to output though their hidden layers and activation functions. The scaling of the output of this network into proper ranges depends on the number of hidden layers, neurons in the hidden layers.

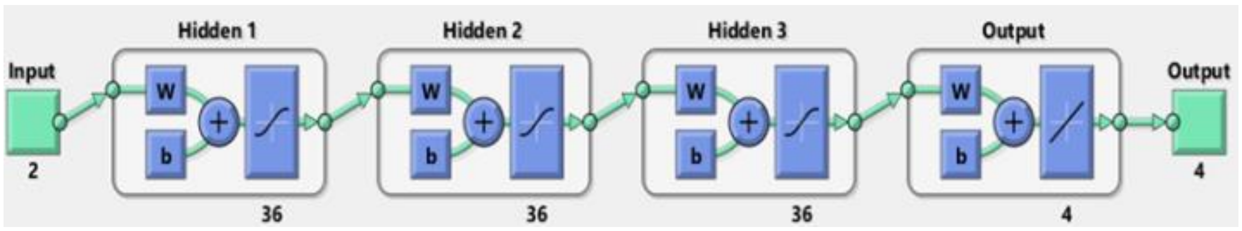


**Figure 3.** FFNN Architecture with three hidden layers and the number of neurons in each layer is 36.

## Results and Discussion

In this work, Neural Network Toolbox™ of MATLAB (Version 2013a) has been used. The generated data is divided into three categories: training data, validation data and test data with ratio 75:15:10 respectively. The performance graph is counted in terms of mean square error (MSE). The number of input layer nodes is two which describes the input features and the number of output layer nodes is four which describes the target (output) classes. The activation (transfer) function in the hidden layers is tan sigmoid and in the output the activation function is pure linear (by default). The number of epochs (iteration) is fifteen.

Now, the performance of this network is evaluated according to increasing the neuron's number in the hidden layer. The measured results and the obtained graph with the increasing neuron numbers in the hidden layer (only one hidden layer) are shown in Figure 4.

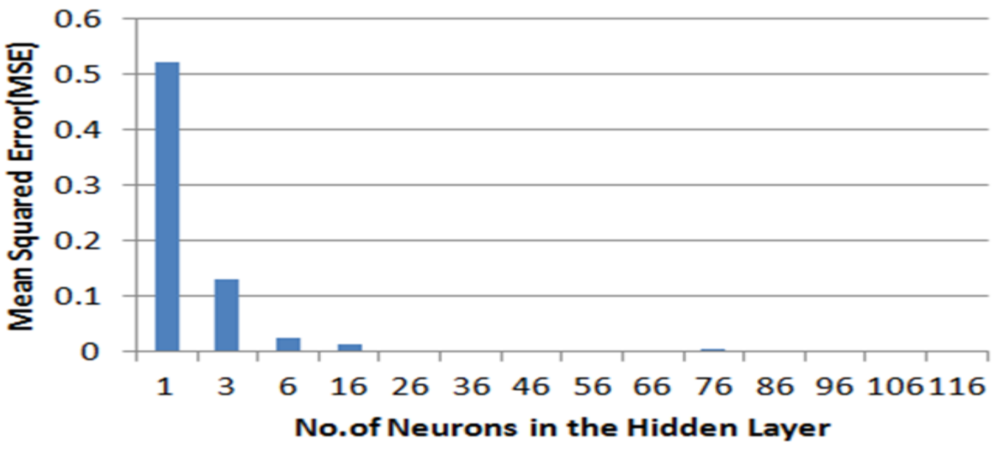


**Figure 4.** The effect of increasing the neuron's number in the hidden layer with respect to MSE.

From Figure 4, it is seen that with increasing the neuron's number the MSE is decreased exponentially and after a certain number of neurons (16 neurons) in hidden layer the MSE is almost stable in spite of increasing the neuron's number in the hidden layer. Finally, it is said that for this network it is better to use the neuron's number in the hidden layer more than 16 but so more the neuron's number in the hidden layer the network takes more time to give the output though MSE is constant. So, considering all things, 26 neurons are chosen in the hidden layer for testing the effect of increasing the number of hidden layers that is shown in Figure 5.

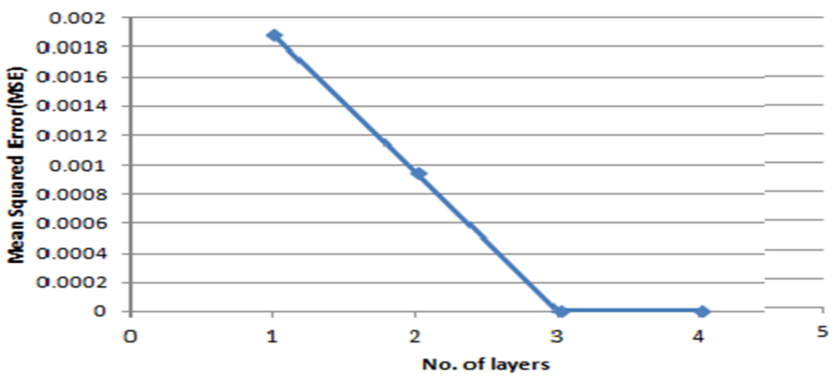
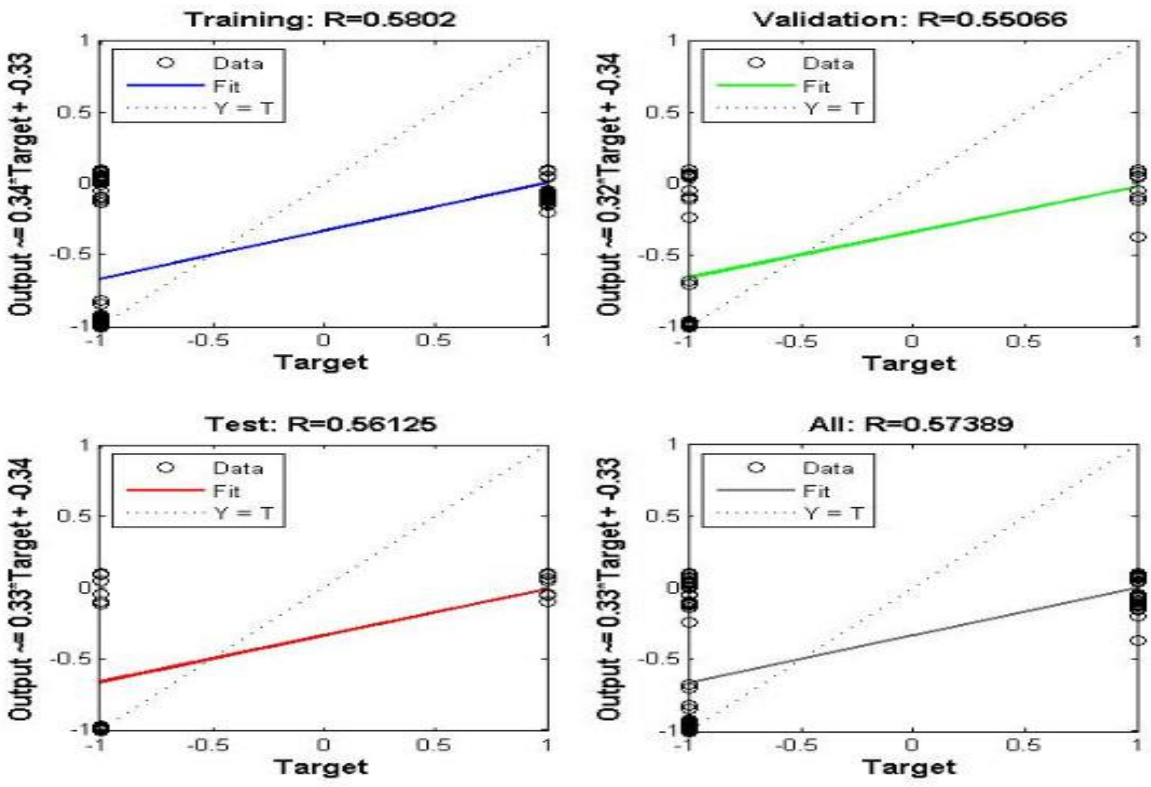


**Figure 5.** The effect of increasing the neuron's number in the hidden layer with respect to MSE.

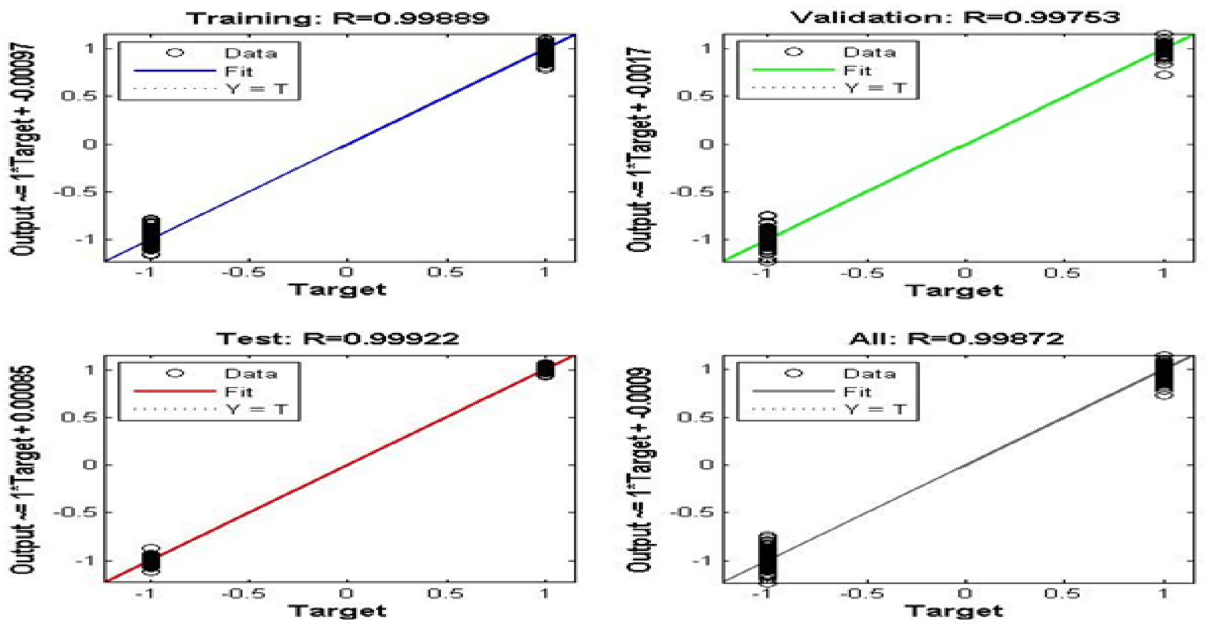
It is observed from Figure 5 that with increasing the number of hidden layers in the network the MSE is decreased linearly and after a certain number of hidden layers (three) the MSE is almost stable in spite of increasing the number of hidden layers. So, it is easily said that with increasing the number of hidden layers contribute to the enhancement of the network performance. The best performance is achieved when the numbers of hidden layers are three because after increasing the number of hidden layers from three the network takes more time to execute though MSE is constant.

The performance of this network is also verified according to increasing the neuron's number in the hidden layers and the number of hidden layers by the regression plot. For the best fitting of data by the network, the intended value of R is equal to 1 and for the worst fit R is equal to 0. From Figure 6, it is observed the effect of increasing the neuron's number in case of only one hidden layer by regression plot.

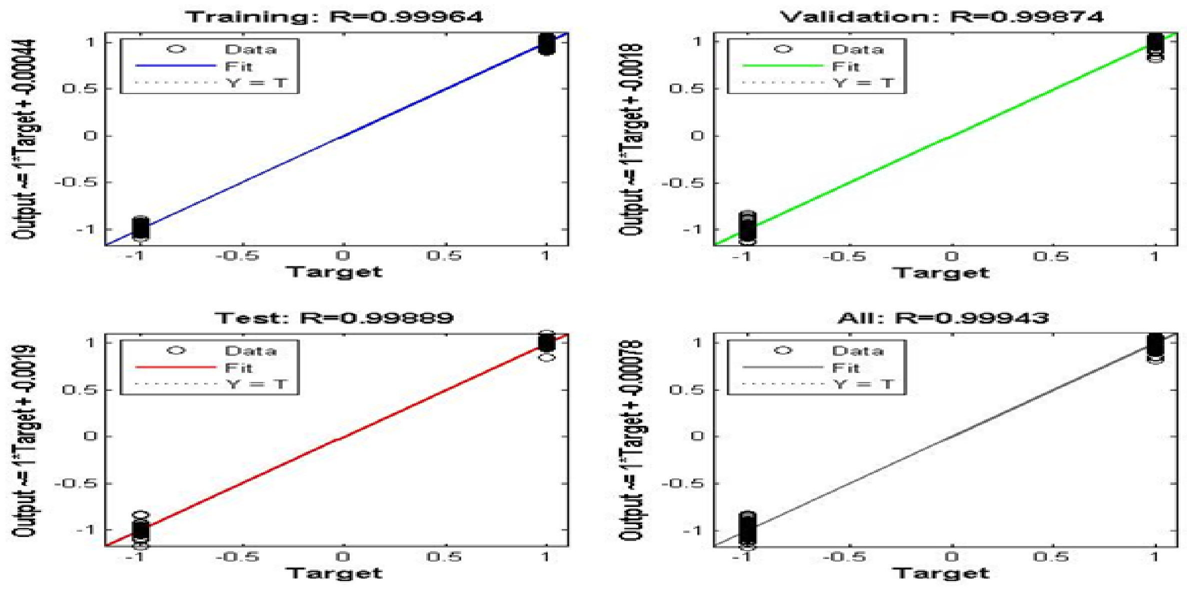
Figure 6(a) gives the regression plot for one neuron. Here, the value of R is equal to 0.5802, 0.56125, 0.55066 and 0.57389 for the training, test, validation and all data respectively. It is clear from this figure that the value of R is hugely drifted from 1. So, it is not better to use one neuron in the hidden layer because for this kind of designed network the classification will not work properly. But from Figure 6(b), it is also observed that the value of R is becoming tends to 1 and this is equal to 0.99889, 0.99922, 0.99763 and 0.99872 for the training, test, validation and all data respectively. Figure 6(c), it is showed that the value of R is equal to 0.99954, 0.99889, 0.99874 and 0.99943 for the training, test, validation and all data respectively that means the value of R for the all the cases is almost same like Figure 6(b).



(a)

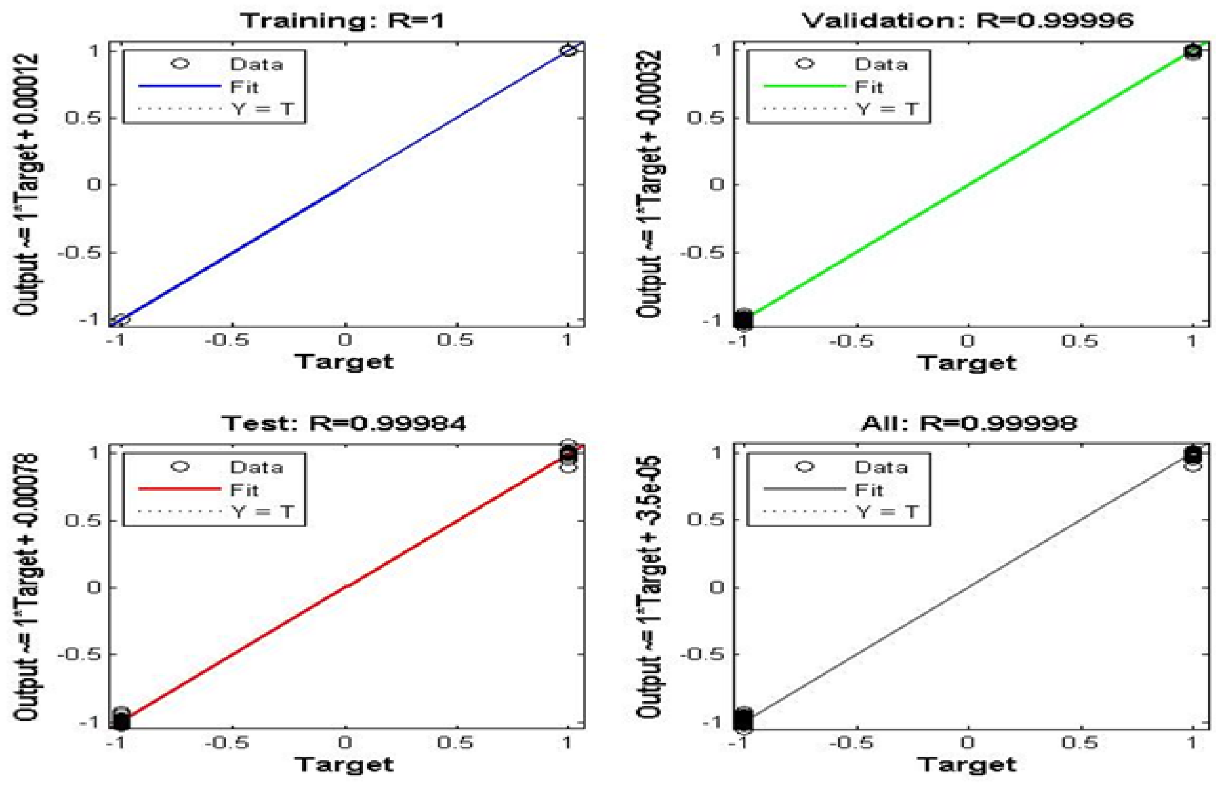


(b)

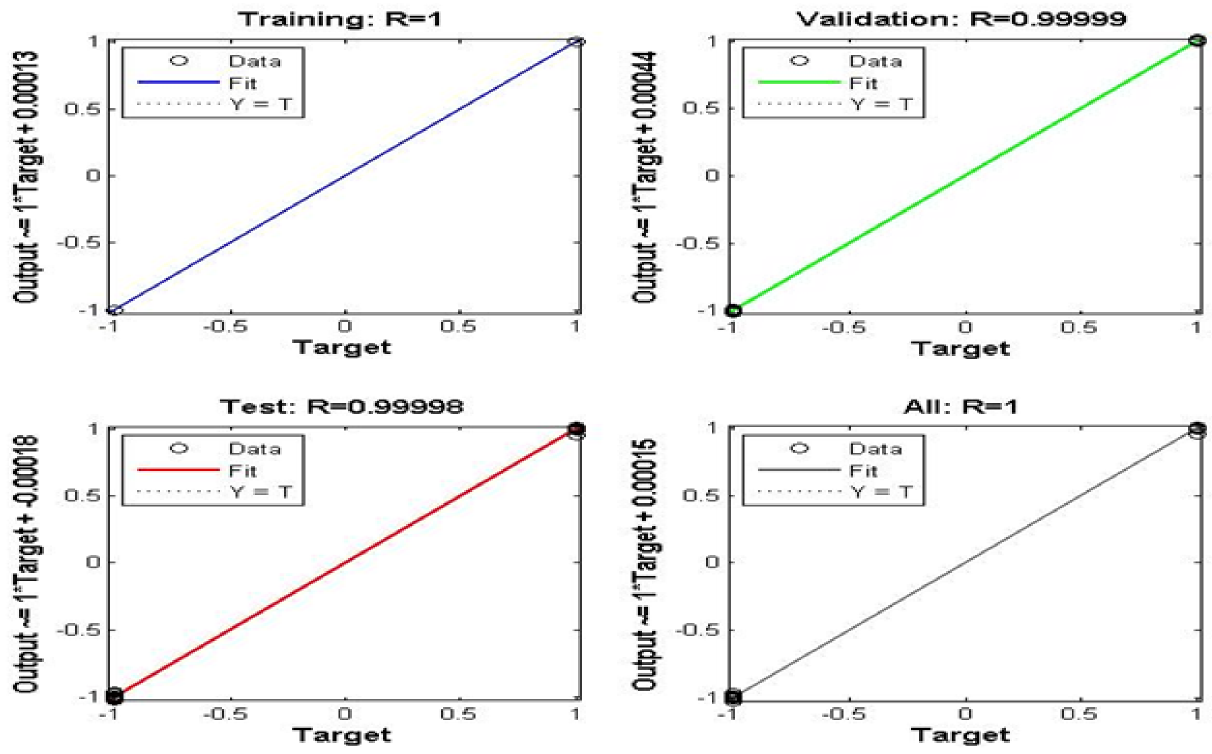


(c)

**Figure 6.** Regression plot for (a) one neuron, (b) 16 neurons, and (c) 26 neurons in case of only one hidden layer.



(a)



(b)

**Figure 7.** Regression plot for (a) two hidden layers, (b) three hidden layers in case of 26 neurons in each layer.

Due to this situation, it is suggested that for this network it is better to use the neuron's number in the hidden layer more than 16 but so more the number of neurons in the hidden layers the network takes more time to execution. So, it is said that 26 neurons in the hidden layer are better for designing this network. Figure 6(c) shows the regression plot for 26 neurons in the only one hidden layer. Figure 7 shows the regression plot for two and three hidden layers where 26 neurons in each hidden layer.

From Figure 7(a), it is observed that the value of R is increased in each case compare to Figure 6(c). This indicates that the network has the better fitting for the data for two hidden layers compare to single hidden layer. From Figure 7(b), it is also shown that the value of R is also increased in each case compare to Figure 7(a). This also indicates that the network has the better fitting for the data for three hidden layers compare to two hidden layers. So from this interpretation, it is said that with increasing the number of hidden layers contribute to the enhancement of performance of this network but so more the number of hidden layers in the network make it complex and takes more time to execute.

From the above illustration, it is obvious that three hidden layers and 26 hidden neurons are better for designing the classifier of FFNNs for this kind of input data features and from the studied literature it is found that two hidden layers and neurons in the hidden layers is equal to the input features are the best choice in most applications.

## Conclusion

From the result and discussion, it is clear that for getting well performance with less training time and complexity form this network it is better to use three the hidden layers and 26 neurons in hidden layers for designing the classifier of this network for this kind of input data features. As increasing the number of hidden layers and neurons in the hidden layers gives the less MSE and better accuracy to a great extent but neural network became more complex and takes more time to execute. So a compromising is taken to design this network. In this study, some methods for selecting the number of hidden layers and the number of hidden nodes was also found from the studied literature and finally from our simulation results and studied literature, it is said that the number of hidden layers and neurons number in the hidden layers are application specific. This concept may also be used in other kinds of ANNs and can be applied in aerospace, banking, defense, entertainment, insurance, oil and gas, speech, securities, telecommunication others for designing the classifiers, especially in the biomedical field for the automatic diagnosis of patients at different conditions more efficiently. As a result, it can help the doctors to take the appropriate decision about the diseases at different critical conditions of patients and reduce the diagnosis error and time as well as cost.

## References

Ahmed, K. & Noureldien, A. (2014). Determining the efficient structure of feed-forward neural network to classify breast cancer dataset. *International Journal of Advanced Computer Science and Applications*, 5(12), 87-90.

Amardeep, R. & Swamy, K. T. (2017). Training feed forward neural network with back propagation algorithm. *International Journal of Engineering and Computer Science*, 6(1),19860-19866.

Angelini, E., di Tollo, G. & Roli, A. (2008). A neural network approach for credit risk evaluation. *Quarterly Review of Economics and Finance*, *Elsevier,* 48(4), 733–755.

Benardos, P. G. & Vosniakos, G. C. (2007). Optimizing feedforward artificial neural network architecture. *Engineering Application of Artificial Intelligence*, 20, 365-382.

Benediktsson, J., Swain, P. & Ersoy, O. (1990). Neural network approaches versus statistical methods in classification of multisource remote sensing data. IEEE *Transactions on Geoscience and Remote Sensing*, 28(4), 540–552.

Cybenko, G. (1989). Approximation by super-positions of a sigmoidal function. *Math. Control Signals System*, 2, 303-314.

Danaher, S., Datta, S., Waddle, I. & Hackney, P. (2004). Erosion modelling using Bayesian regulated artificial neural networks. *Wear,* 256(9-10), 879–888.

Engelbrecht, A. P. (2007). *Computational Intelligence: An Introduction - second edition.* John Wiley & Sons Ltd.

Hagan, M. T., Demuth, H. B., & Beale, M. (1995). *Neural Network Design. USA, Boston, Mass,* PWS Publishing Company.

Han, D. Li, M. & Wang, J. (2012). Chaotic time series prediction based on a novel robust echo state network. *IEEE Transactions on Neural Networks and Learning Systems*, 23(5), 787– 799.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. MacMillan, New York, USA.

Hornik, K., Stinchcombe, M. & White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5), 551–560.

Huang, G. B. & Babri, H. (1997). General approximation theorem on feed-forward networks. 1997 In International Conference on Information, Communications and Signal Processing, ICICS ’97 (pp.698-702), Singapore.
Huang, G. B. (2003). Learning capability and storage capacity of two-hidden-layer feed-forward networks. *IEEE Transactions on Neural Networks*,14(2), 274–281.

Jaeger, H. (2002). *Tutorial on Training Recurrent Neural Networks, Covering BPPT, RTRL, EKF and the “Echo State Network” Approach (pp. 48).* GMD Report 159, German National Research Center for Information Technology.

Jaeger, H. (2007). Echo state network. *Scholarpedia*, 2(9), article no. 2330

Kurkova, V. (1992). Kolmogorov’s theorem and multilayer neural networks. *Neural Networks*, 5, 501–506.

Lin, T., Horne, B. G. Tiˇno, P. & Giles, C. L. (1996). Learning long term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6),1329–1338.

Lin, Y. Y., Chang, J. Y., Pal, N. R., and Lin, C. T. (2013). A mutually recurrent interval type-2 neural fuzzy system (MRIT2NFS) with self-evolving structure and parameters, Fuzzy Systems. IEEE Transactions on 21.3, 2013: 492-509.

Mas, J.F. & Flores, J. J. (2008). The application of artificial neural networks to the analysis of remotely sensed data. *International Journal of Remote Sensing*, 29(3), 617–663.

Naguib, R. N. & Sherbet, G. V. (2001). *Artificial neural networks in cancer diagnosis, prognosis, and patient management (pp. 212-213).* New York, CRC Press LLC.

Nielsen, R. H. (1987). Kolmogorov’s mapping neural network existence theorem. 1987 In IEEE First Annual International Conference on Neural Networks (pp.11-13), 3.

Pacelli, V. & Azzollini, M. (2011). An artificial neural network approach for credit risk management. *Journal of Intelligent Learning Systems and Applications*, 3(2), 103–112.

Panchal, F. & Panchal, M. (2015). Optimizing number of hidden nodes for artificial neural network using competitive learning approach. *International Journal of Computer Science and Mobile Computing*, 4(5), 358 – 364.

Rani, N. & Vashisth, S. (2016). Brain tumor detection and classification with feed forward back propagation neural network. *International Journal of Computer Applications*, 146(12), 1-6.

Rodan, A. & Tiˇno, P. (2011). Minimum complexity echo state network. *IEEE Transactions on Neural Networks,* 22(1), 131–144.

Rodan, A., Faris, H. & Alqatawna, J. (2016). Optimizing feed-forward neural networks using biogeography based optimization for E-mail spam identification. *Int. J. Communications, Network and System Sciences*, 9, 19-28.

Stathakis, D. (2009). How many hidden layers and nodes? *International Journal of Remote Sensing* (Taylor & Francis Publishing), 30(8), 2133–2147.

Valiente, A. D. C., Sequera, J. L. C., Martinez, A. C., Pulido, J. M. G. & Martinez, J. M. G. (2017). An artificial neural network for analyzing overall uniformity in outdoor lighting systems. *Energies*, 10(2), 1-18.

Wilamowski, B. M. (2009). Neural network architectures and learning algorithms. *IEEE Industrial Electronics Magazine*, 3(4), 56–63..

**How to cite this paper:**
Ullah, H., Kiber A., Huq, A. & Bhuiyan, M.A.S. (2018). Computing the Performance of FFNN for Classifying Purposes. *Malaysian Journal of Applied Sciences, 3*(2), 8-20.