International Journal of Computer Theory and Engineering, Vol. 3, No. 2, April 2011
ISSN: 1793-8201

Behaviour Analysis of Multilayer Perceptrons with Multiple Hidden Neurons and Hidden Layers

Gaurang Panchal[1], Amit Ganatra[2], Y P Kosta[3] and Devyani Panchal[4]

Abstract—**The terms "Neural Network" (NN) and "Artificial Neural Network" (ANN) usually refer to a Multilayer Perceptron Network. It process the records one at a time, and "learn" by comparing their prediction of the record with the known actual record. The problem of model selection is considerably important for acquiring higher levels of generalization capability in supervised learning. This paper discussed behavioral analysis of different number of hidden layers and different number of hidden neurons. It's very difficult to select number of hidden layers and hidden neurons. There are different methods like Akaike's Information Criterion, Inverse test method and some traditional methods are used to find Neural Network architecture. What to do while neural network is not getting train or errors are not getting reduced. To reduce Neural Network errors, what we have to do with Neural Network architecture. These types of techniques are discussed and also discussed experiment and result. To solve different problems a neural network should be trained to perform correct classification..**

Index Terms—**Back Propagation; Neural Network; Training; Testing; Weights.**

I. Introduction To Neural network

Neural networks are most effective and appropriate artificial intelligence technology for pattern recognition. Superior results in pattern recognition can be directly applied for business purposes in forecasting, classification and data analysis [1]. This new approach gives an extra advantage in solving "real-world" problems in business and engineering. However, to bring proper results, neural networks require correct data pre-processing, architecture selection and network training. In general terms, an artificial neural network consists of a large number of simple processing units, linked by weighted connections. Each unit receives inputs from many other units and generates a single output. The output acts as an input to other processing units. An MLP is a network of simple *neurons* called *perceptrons*. The basic concept of a single perceptron was introduced by Rosenblatt in 1958. The perceptron computes a single *output* from multiple real-valued *inputs* by forming a linear combination according to its input *weights* and then possibly putting the output through some nonlinear activation function. There are really two decisions that must be made regarding the hidden layers: how many hidden layers to actually have in the neural network and how many neurons will be in each of these layers. We will first examine how to determine the number of hidden layers to use with the neural network. Problems that require two hidden layers are rarely encountered. However, neural networks with two hidden layers can represent functions with any kind of shape. There is currently no theoretical reason to use neural networks with any more than two hidden layers. In fact, for many practical problems, there is no reason to use any more than one hidden layer.

Manuscript received March 28, 2011

Department of Computer Engineering, Charotar Institute of Technology (Faculty of Technology and Engineering),
Charotar University of Science and Technology, Changa, Anand-388 421, INDIA
[1] gaurangpanchal.ce@ecchanga.ac.in, [2] amitganatra.ce@ecchanga.ac.in, [3] ypkosta.adm@ecchanga.ac.in, [4] devyanipanchal.it@ecchanga.ac.in

II. Introduction To Multilayer Perceptrons

Multilayer perceptrons (MLPs) are feed forward neural networks trained with the standard back propagation algorithm. They are supervised networks so they require a desired response to be trained. They learn how to transform input data into a desired response, so they are widely used for pattern classification. With one or two hidden layers, they can approximate virtually any input-output map. They have been shown to approximate the performance of optimal statistical classifiers in difficult problems. Most neural network applications involve MLPs.

This is perhaps the most popular network architecture in use today. The units each perform a biased weighted sum of their inputs and pass this activation level through a transfer function to produce their output, and the units are arranged in a layered feed forward topology. The network thus has a simple interpretation as a form of input-output model, with the weights and thresholds (biases) the free parameters of the model. Such networks can model functions of almost arbitrary complexity, with the number of layers, and the number of units in each layer, determining the function complexity. Important issues in Multilayer Perceptrons (MLP) design include specification of the number of hidden layers and the number of units in these layers.

The number of input and output units is defined by the problem (there may be some uncertainty about precisely which inputs to use, a point to which we will return later. However, for the moment we will assume that the input variables are intuitively selected and are all meaningful). The number of hidden units to use is far from clear. As good a starting point as any is to use one hidden layer, with the number of units equal to half the sum of the number of input

and output units. Again, we will discuss how to choose a sensible number later.

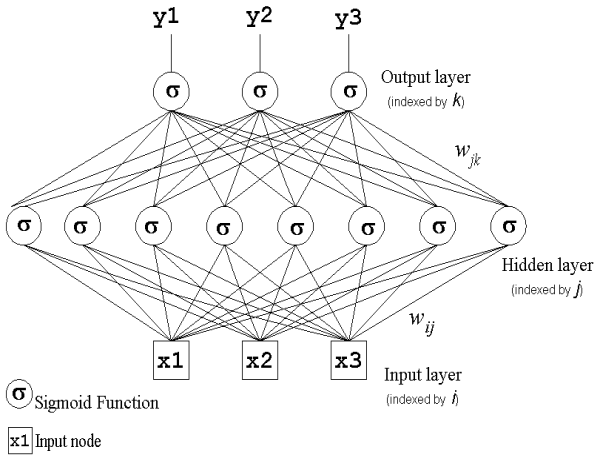

Figure 1. Architecture of Multilayer Perceptron

Multilayer perceptrons (MLPs) are layered feed forward networks typically trained with static back propagation. Here you simply specify the number of hidden layers. These networks have found their way into countless applications requiring static pattern classification. Their main advantages are that they are easy to use, and that they can approximate any input/output map. The key disadvantages are that they train slowly, and require lots of training data (typically three times more training samples than network weights).

III. Training Multilayer Perceptron Networks

There are several issues involved in designing and training a multilayer perceptron network:

- Selecting how many hidden layers to use in the network.
- Deciding how many neurons to use in each hidden layer.
- Finding a globally optimal solution that avoids local minima.
- Converging to an optimal solution in a reasonable period of time.
- Validating the neural network to test for over fitting.
-

Once the number of layers, and number of units in each layer, has been selected, the network's weights and thresholds must be set so as to minimize the prediction error made by the network. This is the role of the training algorithms. The historical cases that you have gathered are used to automatically adjust the weights and thresholds in order to minimize this error. This process is equivalent to fitting the model represented by the network to the training data available. The error of a particular configuration of the network can be determined by running all the training cases through the network, comparing the actual output generated with the desired or target outputs. The differences are combined together by an error function to give the network error. The most common error functions are the sum squared error (used for regression problems), where the individual errors of output units on each case are squared and summed together, and the cross entropy functions (used for maximum likelihood classification).

IV. Hidden Layer Selection

For nearly all problems, one hidden layer is sufficient. Two hidden layers are required for modeling data with discontinuities such as a saw tooth wave pattern. Using two hidden layers rarely improves the model, and it may introduce a greater risk of converging to a local minima. There is no theoretical reason for using more than two hidden layers.

V. Hidden Neurons Selection in Hidden Layer

Deciding the number of neurons in the hidden layers is a very important part of deciding your overall neural network architecture. Though these layers do not directly interact with the external environment, they have a tremendous influence on the final output. Both the number of hidden layers and the number of neurons in each of these hidden layers must be carefully considered. Using too few neurons in the hidden layers will result in something called under fitting. Under fitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set.

Using too many neurons in the hidden layers can result in several problems. First, too many neurons in the hidden layers may result in over fitting. Over fitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers.

There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

These three rules provide a starting point for you to consider. Ultimately, the selection of architecture for your neural network will come down to trial and error. You do not want to start throwing random numbers of layers and neurons at your network. To do so would be very time consuming.

The best number of hidden units depends in a complex way on:

- The numbers of input and output units
- The number of training cases

- The amount of noise in the targets
- The complexity of the function or classification to be learned
- The architecture o the type of hidden unit activation function
- The training algorithm

VI. Different Approaches for Hidden Neurons Selection

A. Simple Method

There is a simple method to find out Neural Network hidden Neurons. Assume a Back Propagation Neural Network Configuration is l-m-n. Here l is input neurons, m is hidden neurons and n is output layers. If we have two input and two output in our problem than we can take same number of hidden neurons. So our configuration becomes 2-2-2. (l=2 input neurons=2 hidden neurons and n=2 output neurons.)

Total numbers of weights to be determined are

$$Weights = (1+n)*m \quad (1)$$

B. Based on Hopfiled Neural Network

The Hopfield network consists of a set of N interconnected neurons which update their activation values asynchronously and independently of other neurons. All neurons are both input and output neurons. The Hopfield neural network is perhaps the simplest of neural networks. The Hopfield neural network is a fully connected single layer auto associative network. This means it has one single layer, with each neuron connected to every other neuron. In this chapter we will examine a Hopfield neural network with just four neurons. This is a network that is small enough that it can be easily understood, yet can recognize a few patterns. A Hopfield network with connections we will build an example program that creates the Hopfield network shown in Figure 2.6. A Hopfield neural network has every Neuron connected to every other neuron. This means that in a four Neuron network there are a total of four squared or 16 connections. However, 16 connections assume that every neuron is connected to itself as well. This is not the case in a Hopfield neural network, so the actual number of connections is 12.From this concepts we can take same number of hidden neurons and input neurons.

C. Akaike's Information Criterion (AIC)

Akaike's information criterion, developed by Hirotsugu Akaike under the name of "an information criterion" (AIC) in 1971 and proposed in Akaike (1974), is a measure of the goodness of an estimated statistical model. It is grounded in the concept of entropy, in effect offering a relative measure of the information lost when a given model is used to describe reality and can be said to describe the tradeoff between bias and variance in model construction. The AIC is not a test on the model in the sense of hypothesis testing; rather it is a tool for model selection. Given a data set, several competing models may be ranked according to their AIC, with the one having the lowest AIC being the best. From the AIC value if top three models are in a tie and the rest are far worse, but one should not assign a value above which a given model is 'rejected'. The AIC is a basis of comparison and selection among several statistical models. As we all know the goodness of parameters of a model can be calculated by the expected log likelihood, means the larger the expected log likelihood is better explanation. In looking at the relationship between the bias and the number of free parameters of a model [1], it is found that,

(Maximum log likelihood of a model) – (number of free parameters of the model)

It is an asymptotically unbiased estimator of the mean expected log likelihood. AIC estimator of Kullback –Leibler information is

AIC = -2 * (maximum log likelihood of the model) + 2 * (number of free parameters of the model).
In the general case, the AIC [5] is,

$$AIC = -2*\ln(likelihood) + 2*k \quad (2)$$

Where ln is the natural logarithm, k is the number of parameters in the statistical model and RSS is the residual sums of squares (Calculation of RSS value is discussed later in this paper). AIC can also be calculated using residual sums of squares [5] from regression

$$AIC = n*\ln(RSS/n) + 2*K \quad (3)$$

Where n is the number of data points (observations). AIC requires a bias-adjustment small sample sizes. If ratio of n/K < 40 then uses bias adjustment

$$AIC = -2*\ln(L) + 2*K + (2*K*(K+1))/(n-K-1) \quad (4)$$

For example, consider 3 candidate models for the growth model, their RSS values, and assume n = 100 samples in the data.Table1 shows the Calculation of AIC values for different Models. AIC is calculated using different RSS values and different no of free parameters. Lower AIC is better for model.

$$RSS = \sum \varepsilon_i^2 = 1.13 \quad (5)$$

Where RSS is Residual sum of Square and E is error.

The best model is determined by examining their relative distance to the "truth". The first step is to calculate the difference between lowest AIC model and the others as

$$\Delta i = AICi - \min AIC \quad (6)$$

Where Δi is the difference between the AIC of the individual models and min AIC is the minimum AIC value of all models [5]. The smallest value of AIC is –130.21(using equation 5). Thus the Δi is show in table1.

TABLE I. CALCULATION OF ΔI

K	RSS	AICc	Δ i
4	25	-130.21	0
3	26	-114.15	1.75
3	27	-98.73	5.53

To quantify the plausibility of each model as being the best approximating, we need an Estimate of the likelihood of our model given our data

$$L(Model \mid data)$$

Interestingly, this proportional ( ) to the exponent of

(-0.5*Δi) so that

$$L(Model \mid data) \alpha \exp(-0.5 * \Delta i) \quad (7)$$

The right hand side of above is known as the relative likelihood of the model, given the data. A better means of interpreting the data is to normalize the relative likelihood [5] values as

$$\sum i = \exp(-0.5 * \Delta i) / \sum_{i}^{R} \exp(-0.5 * \Delta i) \quad (8)$$

TABLE II. EXPONENT OF DELTA

K	RSS	AICc	Δ i	exp(-0.5* Δ i)
4	25	-130.21	0	1
3	26	-128.46	1.75	0.4166
3	27	-124.68	5.52	0.0631
				Sum = 1.4798

The sum of the relative likelihoods is 1.4798, so we obtain the Akaike weights for each by dividing the relative likelihood by 1.4798.

TABLE III. AKAIKE'S WEIGHTS FOR DIFFERENT MODELS

K	RSS	AICc	Δ i	Wi
4	25	-130.21	0	0.6758
3	26	-128.46	1.75	0.2816
3	27	-124.68	5.52	0.0427

Where W_i are known as Akaike weights for model I and the denominator is simply the sum of the relative likelihoods for all candidate models. For example, using the earlier values from the 3 growth models:

For the above example, the first model is (0.6758/0.2816) = 2.4 times more likely to be the best explanation for growth compared to second Model only and (0.6758/0.0427) = 15.8 times more likely than third model only.

As a general rule of thumb, the confidence set of candidate models (analogous to a confidence interval for a mean estimate) include models with Akaike weights that are within 10% of the highest, which is comparable with the minimum cut-off point (i.e., 8 or 1/8) suggested by Royall (1997) as a general rule-of-thumb for evaluating strength of evidence.

For the above example, this would include any candidate model with a value greater than (0.6758*0.10) = 0.0676. Thus, we would probably exclude the third model only from the model confidence set because its weight, 0.0427<0.0676

D. Inverse Test Error Method

In mathematics, the error function (also called the Gauss error function) is a special function (non-elementary) of sigmoid shape which occurs in probability, statistics, materials science, and partial differential equations. How to select neurons using Inverse test error method is discussed later.

The error function is used in training the network and in reporting the error. The error function used can have a profound effect on the performance of training algorithms.

E. Depend on Neural Network Error

If neural network is not producing correct output at that time we have to increase number of hidden neurons. Also if neural network is not giving less error at time we have to increase number of hidden layers.

F. Depend on Neural Network Training

If neural network is getting train at that time we have to increase number of hidden neurons.

G. Depend on Neural Network Output

If neural network is not giving predicated output at that time we have to increase number of hidden neurons/Layers. To creating best neural network architecture we have to increase number of hidden neurons by 2 or 3 increment.

H. Hidden Layers and MSE

Hidden layers can also be finding out by following relationship,

$$N_{hl} \alpha N_e \quad (9)$$

Where N_e is number of epochs and N_{hl} is number of hidden layers. So,

$$N_e \alpha \frac{1}{Min(MSE)} \quad (10)$$

So,

$$N_{hl} \alpha \frac{1}{Min(MSE)} \quad (11)$$

VII. PROBLEM STATEMENT

Here we have considered employee retention Problem for behavioral analysis of multilayer perceptrons and hidden neurons. In this problem there are 17 inputs and one output is Employee Retention Probability. This problem is analyzed with different methods as mention before. In next section we will see all the methods one by one.

VIII. Behavioral analysis

A. As per First method the network configuration becomes l-m-n (l is input neurons is hidden neurons and n is output neurons).Here l=17, m=17 and n=1 as per the method.

TABLE IV. HIDDEN LAYER=1 AND HIDDEN NEURONS=17

	Training	Validation
CCR,%:	100	-
Network Error:	0.000298	0
Iteration:	501	
Training Speed:,Iter/sec:	47.264175	
Architecture:	[17-17-1]	

TABLE V. HIDDEN LAYER=2 AND HIDDEN NEURONS=8 IN EACH

	Training	Validation
CCR,%:	100	
Network Error:	0.000298	0
Iteration:	501	
Training Speed:,Iter/sec:	52.187524	
Architecture:	[17-8-8-1]	

Table number 4 the architecture with 17 hidden neurons with one hidden layer which gives 0.000298 network error while table 5 shows the neural network architecture with two hidden layers and 8 hidden neurons in each layer though we are getting same network error.

B. Using Akaike's Information Criterions

The Akaike's information criterion is most successful method to find neural network architecture. By using this method we can find number of hidden neurons for given problem. The method is discussed in previous section. Lower AIC is better for model.

TABLE VI. AIC CALCULATION

ID	Architecture	# of Weight	Fitness	AIC
1	**[17-3-1]**	**58.00**	**9164.053**	**-9164.053**
2	[17-43-1]	818.00	7644.053	-7644.053
3	[17-27-1]	514.00	8252.053	-8252.053
4	[17-17-1]	324.00	8632.053	-8632.053
5	[17-11-1]	210.00	8860.053	-8860.053
6	[17-7-1]	134.00	9012.053	-9012.053
7	[17-9-1]	172.00	8936.053	-8936.053
8	[17-5-1]	96.00	9088.053	-9088.053
9	[17-6-1]	115.00	9050.053	-9050.053

We have started searching neural network architecture with hidden neurons 2 to 30 and incrementing hidden neurons by 3.We get better result in first architecture [17-3-1] with lowest AIC

TABLE VII. HIDDEN LAYER=1 AND HIDDEN NEURONS=3

	Training	Validation
CCR,%:	100	
Network Error:	0.000303	0
Iteration:	501	
Training Speed:,Iter/sec:	131.84	
Architecture:	[17-3-1]	

C. Using Inverse Test Error Method

TABLE VIII. HIDDEN LAYER=1 AND HIDDEN NEURONS=2

	Training	Validation
CCR,%:	100	
Network Error:	0.000318	0
Iteration:	501	
Training Speed:,Iter/sec:	125.25	
Architecture:	[17-2-1]	

By using this method we are getting more network errors but time taken to execute is less than the previous methods.

D. With different Hidden Neurons and Layers

TABLE IX. HIDDEN LAYER=2 AND HIDDEN NEURONS=4 IN EACH

	Training	Validation
CCR,%:	100	
Network Error:	0.000287	0
Iteration:	501	
Training Speed:,Iter/sec:	83.5	
Architecture:	[17-4-4-1]	

TABLE X. HIDDEN NEURONS IN EACH LAYERS IS 8, 4 AND 2 RESPECTIVELY

	Training	Validation
CCR,%:	100	
Network Error:	0.000304	0
Iteration:	501	
Training Speed:,Iter/sec:	55.66	
Architecture:	[17-8-4-2-1]	

From result of table 4 and 5, we can say that as soon as number of unnecessary layers increase than network error will increase.

IX. Experiment and Results

We have taken the employee retention problem with nine inputs like experience, age, qualification etc, and we are getting following result.

TABLE XI. CRR WITH DIFFERENT NN ARCHITECTURE

# Hidden Layer	Architecture	CRR %
1	[9-1-1]	95.65
1	[9-2-1]	96.38
1	[9-3-1]	96.38
1	[9-4-1]	95.65
1	[9-5-1]	95.65
1	[9-6-1]	95.65
1	[9-7-1]	96.33
1	[9-8-1]	95.65

1	[9-10-1]	97.83
1	[9-15-1]	95.65
2	[9-5-51]	96.38
3	[9-5-5-1]	97.1

From the above table we can say that as soon as we increase the number of hidden layers or number of hidden neurons we are getting better results or Classification Rate will increase and error will reduces. If we analyze the above result in graphical representation, we get better idea. The graphical representation is shown below.

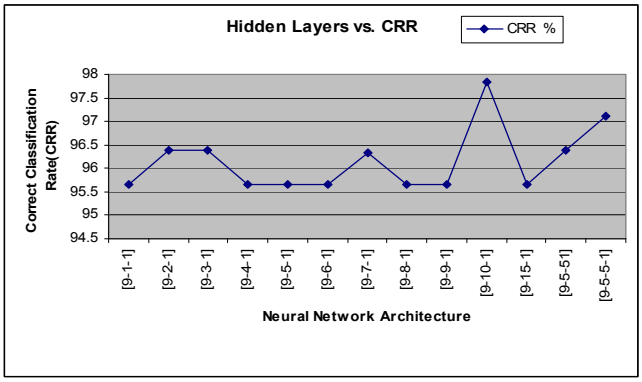

Figure 2. Graphical Representation of Hidden Layers vs. CRR

From the above figure shows that as soon as numbers of hidden neurons and hidden layers increase neural network get better performance. We have discussed various techniques to select number of hidden neurons and number of hidden layers for better neural network result. As we expand the neural network architecture, the training time will also increase.

X. Conclusion

The training process of neural network become slowdown because of number of epoch increased. So if accuracy of the result is critical factor for an application then more hidden layers should be used but if time is major factor at that time single hidden layer should be used.

Acknowledgements

The authors' wishes to thank all the colleagues for their guidance, encouragement and support in undertaking the research work. Special thanks to the Management for their moral support and continuous encouragement.

References

[1] Goldberg, D.E. "Genetic algorithms in search, optimization, and machine learning", Pearson Education, 2001.

[2] R. K. Gupta, A. K. Bhunia, "An Application of real-coded Genetic Algorithm for integer linear programming", AMO-Advanced Modeling and Optimization, Volume 8, Number 1, 2006.

[3] V.srinivas, G.L.Thompson. "Benefit-cost Analysis of coding techniques for the Primal Transportation Algorithm.", Journal of the association for computing machinery, Vol.20, April 1973, pp194-213

[4] Man Mohan, Gupta P.K .Operations research, Methods and Solutions.Reading, Sultan Chand and Sons, 1992

[5] BL Mak, H Sockel – Info. & Mgmt, A confirmatory factor analysis of IS employee motivation and retention, 2001

[6] K. Hornik,M. Stnchcombe and H White,Multilayer Feedforward Networks are Universal Approximators, Neural Network,2:pg359-366,1989

[7] Poonam Garg,Advanced in Computer Science & Engineering, MacMillan Publication,2009.

[8] Vijyalakshmi Pai G.A. & Rajasekaran S. Neural networks, fuzzy logic and genetic algorithms, Synthesis and applications. Reading, Prentice-Hall of India, 2004

[9] S. Kullback and R. A. Leibler, 'On Information and Sufficiency', The Annals of Mathematical Statistics, Vol. 22, No. 1, pp. 79–86, 1951

[10] H. Akaike, 'A new look at the statistical model identification', IEEE Transactions on Automatic Control, Vol. 19, No. 6, pp. 716–723, 1974

[11] H. Linhart and W. Zucchini, Model Selection, John Wiley and Sons, 1986

[12] C. M. Hurvich and C. Tsai, 'Regression and Time Series Model Selection in Small Samples', Biometrika, Vol. 76, pp. 297–307, 1989

[13] J. E. Cavanaugh, 'Unifying the Deriviations for the Akaike and Corrected Akaike Information Criteria', Statistics & Probability Letters, Vol. 33, pp. 201–208, 1997