A DATA CLASSIFIER MODEL BASED ON ARTIFICIAL NEURAL NETWORK USING BACK PROPAGATION ALGORITHM

R. Siva Kumar[1], Dr M.SRIDHAR[2*]

Abstract **-** This paper aims to present the details of Artificial Neural Network and the various issues addressed in developing a classifier model using ANN. A Feed Forward Neural Network trained using Back Propagation algorithm is developed for data classification. The performance of the developed ANN based classifier model has been tested using the Iris and Wine data set available in the UCI machine learning repository. For both the cases, simulation results shows that the proposed neural classifier outperform the conventional classifiers.

Index Terms **-** Data Classifier Model, Artificial Neural Network, A Feed Forward Neural Network, Back Propagation algorithm, Iris and Wine data set.

I. INTRODUCTION

The primary task of any data classifier is to appropriately classify patterns into one of various classes which may or may not be known in advance [1]. Due to their powerful nonlinear function approximation and adaptive learning capabilities, neural networks have drawn great attention in the field of data classification. Data classification is a two step process, in the first step; a model is built describing a predetermined set of data classes and in the second step; the constructed model is used for classification.

This paper reviews the fundamentals of artificial neural networks and demonstrates the suitability of Feed Forward Neural Network for
complex data classification problems. Also the various issues involved in the design of ANN-based classifier model are discussed.

II. ARCHITECTURE OF FEED FORWARD NEURAL NETWORK

Artificial Neural Network (Bishop, 1995 and Haykin, 1999) is an information-processing paradigm inspired by the way the brain processes information. It is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem. Figure 3.1 shows the structure of a three layer Feed Forward Neural Network (FFNN).

The neural network architecture in this class shares a common feature that all neurons in a layer are connected to all neurons in adjacent layers through unidirectional branches. That is, the branches and links can only broadcast information in one direction, that is, the "forward direction"[2]. The branches have associated weights that can be adjusted according to a defined learning rule. Feed forward networks do not allow connections between neurons within any layer of architecture. At every neuron, the output of the linear combiner, that is, the neuron activity level is input to a non linear activation function f (.), whose output is the response of the neuron.

* **Research Scholar Dept of ECE BHARATH UNIVERSITY India Dept of ECE BHARATH UNIVERSITY India**

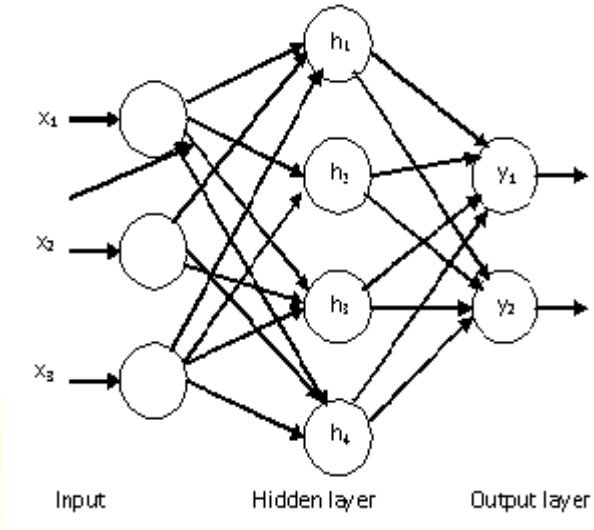

Figure 1. Achitecture of Feed Forward Neural Network

The neurons in the network typically have activity levels in the range [-1, 1], and in some applications the range [0, 1] is used. In Figure 1, actually there are three layers; The first layer in the network does not perform any computations, but only serves to feed the input signal to the neurons of the "second" layer (called the hidden layer), whose outputs are then input to the "third" layer (or the output layer).

The output of the output layer is the network response factor. This network can perform the non linear input/output mapping. In general there can be any number of hidden layers in the architecture; however, from a practical perspective, only one or two hidden layers are typically used. In fact, it can be shown that an MLP that has only one hidden layer, with a sufficient number of neurons, act as a universal approximator of non-linear mapping. In practice, it is very difficult to determine a sufficient number of neurons necessary to achieve the desired degree of approximation accuracy. Hence, trial and error procedure is followed to determine the number of units in the hidden layer.

III. LEARNING IN NEURAL NETWORK

Neural Network models derive their versatility and power primarily from their inherent ability to adapt to changing environments. In the process of adaptation they generate internal models of sampled environmental data. These internal models are represented in variously "structured" weight vectors. Learning algorithms define an architecture-dependent procedure to encode data information into weights to generate these internal models. Learning proceeds by modifying connection strengths. In biological systems learning alters the efficacy of a synapse, both in the amount of neurotransmitter released by a synaptic terminal, and in the physical structure of the axonal-dendritic junction that would allow greater or lesser amount of the neurotransmitter to affect postsynaptic channels. In artificial systems, learning changes the synaptic weights in the model.

Most learning is data driven. The data might take the form of a set of input-output patters derived from a (possibly unknown) probability distribution. Here the output pattern might specify a desired system response for a given input pattern, and the issue of learning

would involve approximating the unknown function as described by the given data. Alternatively, the data might comprise patterns that naturally cluster into some number of unknown classes, and the learning problem might involve generating a suitable classification of the samples.

A number of learning algorithms are available for training and testing the neural networks[3][4]. In this work, Back Propagation learning algorithm is developed for training and testing the feed forward neural network. The details of back propagation algorithm are given in the next section.

IV. BACK PROPAGATION ALGORITHM

Back Propagation learning algorithm (Yegnanarayana, 1999) is a gradient descent method minimizing the mean square error between the actual and target output of a multi layer perceptron[5]. Training the network with back propagation algorithm results in a non-linear mapping between the input and output variables. Thus, given the input/output pairs, the network can have its weights adjusted by the back propagation algorithm to capture the non-linear relationship

The back propagation algorithm consists of the following steps:

Step 1. Initialize weights and offsets

Initialize all weights and node offsets to small random values.

Step 2. Present Input and Desired Output vector

Present continuous input vector x and specify the desired output d. Output vector elements are set to zero values except for that corresponding to the class of the current input.

Step 3. Calculate actual outputs

Calculate the actual output vector y using the sigmoidal nonlinearity

$$f(net_i) = \frac{1}{1 - e^{-net}} \qquad (1)$$

Step 4. Adapt weights

Adjust weights by

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_j x_i' \qquad (2)$$

where x_i' is the output of the node i, η is the learning rate constant and δ_j is the sensitivity of the node j. If node j is an output node, then

$$\delta_j = f'(net_j)(d_j - y_j) \qquad (3)$$

where d_j is the desired output of the node j, y_j is the actual output and $f'(net_j)$ is the derivation of the activation function calculated at net_j. If the node j is an internal node, then the sensitivity is defined as

$$\delta_j = f'(net_j)\sum_k \delta_k w_{jk} \qquad (4)$$

where k sums over all nodes in the layer above the node j. Update equations are derived using the chain derivation rule applied to the LMS training criterion function.

Step 5. Repeat by going to step 2

The simplest stopping criterion is to end when the change in the training criterion function is smaller then some preset value θ. A better approach is a cross-validation technique, stopping training when the error on a separate validation set reaches a minimum. After training, the networks with fixed weights can provide the output for the given input. Once trained, the network can be used as a classifier model for any engineering applications.

V.CLASSIFICATION THROUGH ARTIFICIAL NEURAL NETWORK

In FFNN, the threshold logic neurons can be connected into multiple layers (or fields) in a feed forward fashion[6][7]. This dramatically increases the computational capability of the system as a whole. In general, it is not uncommon to have more than one hidden layer when solving a classification problem. Such a situation arises when one has a number of disjoint regions in the pattern space that are associated into a single class. To understand this better, we make the following observations for binary threshold neurons with two hidden layers apart from the input and the output layer. Each neuron in the first hidden layer forms a hyperplane in the input pattern space.

A neuron in the second hidden layer can form a hyper-region from the outputs of the first layer neurons by performing an AND operation on the hyperplanes. These neurons can thus approximate the boundaries between pattern classes.

The output layer neurons can then combine disjoint pattern classes into decision regions made by the neurons in the second hidden layer by performing logical OR operations.

VI. DESIGN ISSUES

This section discusses a variety of design issues (Satishkumar, 2004 and Sivanandam et al, 2006) that concern the inner workings of the back propagation algorithm.

A. Pattern or Batch Mode Training

The Back Propagation algorithm operates by sequentially presenting the data drawn from a training set to predefined network architecture. There are two choices in implementing this algorithm. Once a data is presented to the network, its gradients are calculated and proceed to change the network weights based on these instantaneous (local) gradient values. This is called pattern mode training. Alternatively, one could collect the error gradients over an entire epoch and then change the weights of the initial neural network in one shot. This is called batch mode training. In this work Batch mode training is followed for training the neural network using Back propagation algorithm.

B. Selection of Network Structure

Both the generalization and approximation ability of a feed forward neural network are closely related to the architecture of the network and the size of the training set. Choosing appropriate network architecture means the number of layers and the number of hidden neurons per layer. Although the back propagation algorithm can be applied to any number of hidden layers, a three-layered network can approximate any continuous function. The problem of selecting the number of neurons in the hidden layers of multilayer networks is an important issue. The number of nodes must be large enough to form a decision region as complex as that required by the problem. And yet the number must not be excessively large so that the weights cannot be reliably estimated by available training data.

In this thesis cross validation approach is used to select appropriate network architecture. The operational procedure of cross validation approach is given below:

- Divide the data set into a training set T_{training} and a test set T_{test}.
- Subdivide T_{training} into two subsets: one to train the network T_{learning}, and one to validate the network $T_{\text{validation}}$.
- Train different network architectures on T_{learning} and evaluate their performance on $T_{\text{validation}}$.
- Select the best network.
- Finally, retrain this network architecture on T_{training}.
- Test for generalization ability using T_{test}.

The number of training pairs also plays an important role during training of the nets. A simple thumb rule is used to find number of training pairs.

C. Weight Initialization

It is important to correctly choose a set of initial weights for the network. Sometimes it can decide whether or not the network is able to learn the training set function. It is common practice to initialize weights to small random values within some interval $[-\varepsilon, \varepsilon]$. Initialization of weights of the entire network to the same value can lead to network paralysis where the network learns nothing-weight changes are uniformly zero. Further, very small ranges of weight randomization should be avoided in general since this may lead to very slow learning in the initial stages.

Alternatively, an incorrect choice of weights might lead to network saturation where weight changes are almost negligible over consecutive epochs. To get the best result the initial weights (and biases) are set to random numbers between -0.5 and 0.5 or between -1 and 1. In general, the initialization of weights (bias) can be done randomly. In this work a specific approach for initializing weights is followed by using Nguyen-widrow (NW) initialization. This approach is based on a geometrical analysis of the response of the hidden neurons to a single input. This method is designed to improve the learning ability of the hidden units.

$$\beta = 0.7(p)^{1/n} = 0.7\sqrt[n]{p} \qquad (5)$$

where,

- n = number of input units,
- p = number of hidden units,
- β = scale factors

D. Momentum Factor

In Back Propagation Network (BPN), the weight change is in a direction that is a combination of current gradient and the previous gradient. This approach is beneficial when some training data are very different from a majority of the data. A small learning rate is used to avoid major disruption of the direction of learning when very unusual pair of training patterns is presented. If the momentum is added to the weight update formula, the convergence is faster. The weights from one or more previous training patterns must be saved in order to use momentum.

For the BPN with momentum, the new weights for training step t+2, is based on t and t+1. It is found that momentum allows the net to perform large weight adjustments as long as the correction proceeds in the same general direction for several patterns. Thus using momentum, the net does not proceed in
the direction of the gradient, but travels in the direction of the combination of the current gradient and the previous direction for which the weight correction is made. The main purpose of the momentum is to accelerate the convergence of error propagation algorithm. This method makes the current weight adjustment with a fraction of the recent weight adjustment.

E.Termination criteria for Training

The motivation for applying back propagation net is to achieve a balance between memorization and generalization; it is not necessarily advantages to continue training until the error reaches a minimum value. The weight adjustments are based on the training patterns. As along as the error for validation decreases training continues. Whenever the error begins to increase, the net is starting to memorize the training patterns. At this point training is terminated.

VII.SIMULATION RESULTS

This section presents the details of the simulation carried out on two datasets IRIS and WINE available in the UCI Machine Learning Repository (Newman et al., 1998). The proposed ANN-based classifier is implemented in MATLAB and executed in a PC with Pentium IV processor with 2.40 GHz speed and 256 MB of RAM. In order to make a comparison with the proposed ANN classifier, two conventional classifiers based on Naives Bayes and k-Nearest Neighbor has been developed and the details are given in tables.

Case (i): Iris Data Classification

Iris data set consists of 150 four dimensional vectors representing 50 plants each of three species iris setosa, iris versicolor and iris virginica. The four input features are sepal length, sepal width, petal length and petal width. Out of 150 samples, 75 data are used for training and 75 data are used for testing. There are four neurons in the input layer that corresponds to the number of input features and three neurons in the output layer and one in each neuron corresponds to any one of the output classes i.e., [0 0 1] corresponds to iris setosa, [0 1 0] corresponds to iris versicolor, and [1 0 0] corresponds to iris virginica. Trial and error procedure was followed to identify the optimal number of hidden nodes. The

network is trained with Least Mean Square (LMS) (Haykin and Deng, 1991) algorithm until it reaches the mean square error of 0.01.

With four hidden nodes, the network took 12.45 seconds to reach the error goal in 1161 epochs. The performance of the network during training is shown in figure 2.

The performance of the network during training is evaluated using independent validation which divides the 75 data considered for training into two set so that 50 data is used for training and 25 data is used for validation. After training, the generalization performance of the network is evaluated with 75 test data. The proposed neural network classifies 74 data correctly which shows 98.66% classification accuracy. During testing the mean square error achieved by the network is 0.0148.

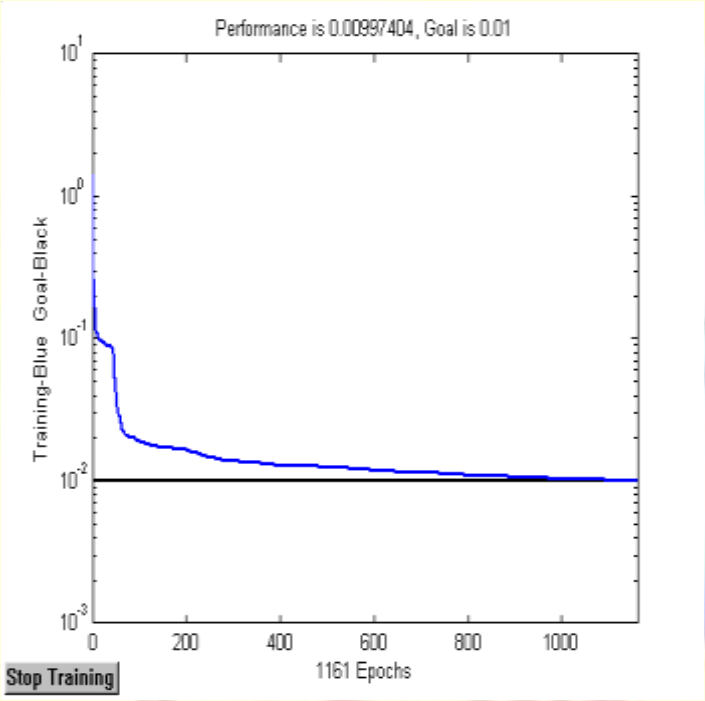

Figure 2 Network training for Iris data

Sl.No	Approach	Correctly Classified Class	Classification Percentage
1	ANN classifier	74	98.66%
2	Naïve Bayes	70	93.33%
3	k-Nearest Neighbor	68	90.66%

Table 1 Performance comparison for Iris data

Table 1 shows the comparison between the performance of the proposed ANN based classifier model with two conventional classifiers based on Naïve Bayes and k-nearest neighbor for Iris data.

Case (ii): Wine Data Classification

Next, an ANN model was developed to classify the wine data. Wine data contains the chemical analysis of 178 wines grown in the same region in Italy but derived from three cultivators which is designated as 3 output classes. The 13 input features are: alcohol, malic acid, ash, alkalinity of ash, magnesium, total phenols, flavanoids, nonflavanoids phenols, proanthocyaninsm color intensity, hue, OD280/OD315 of diluted wines and praline. Among the 13 input features 11 features are continuous and 2 are discrete. Out of 178 samples, 89 data are used for training and 89 data are used for testing

There are thirteen neurons in the input layer that corresponds to the number of input features and three neurons in the output layer. [0 0 1] corresponds to class 1, [0 1 0] corresponds to class 2, and [1 0 0] corresponds to class 3. Trial and error procedure was followed to identify the optimal number of hidden nodes. The network is trained with LM algorithm until it reaches the mean square error of 0.001. With four hidden nodes, the network took 3 seconds to reach the error goal in 213 epochs. The performance of the network during training is shown in figure 3. The mean square error achieved during training 8.9302e-004.

The performance of the network during training is evaluated using independent validation which divides the 89 data considered for training into two set so that 60 data is used for training and 29 data is used for validation. After training, the generalization performance of the network is evaluated with 89 test data. The proposed neural network classifies 88 data correctly which shows 98.87% classification accuracy. During testing the mean square error achieved by the network is 0.0156.

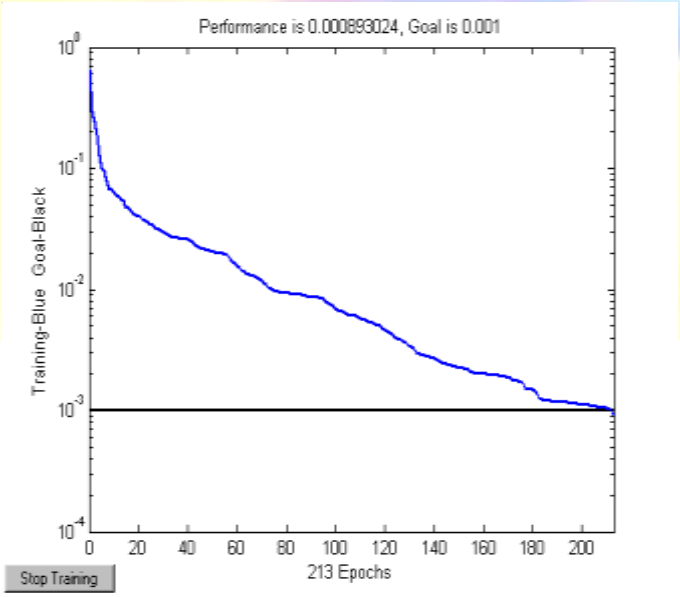

Figure 3. Network Training for wine data

Sl.No	Approach	Correctly Classified Class	Classification Percentage
1	ANN classifier	88	98.87%
2	Naïve Bayes	83	93.25%
3	k-Nearest Neighbor	81	91.01%

Table 2 Performance comparison for wine data

Table 2 shows the comparison between the performance of the proposed ANN based classifier model with two conventional classifiers based on Naïve Bayes and k-nearest neighbor for wine data.

From table 1 and 2, it is clear that the proposed ANN based classifier model has very good classification percentage when compared with two popular conventional classifiers such as Naïve Bayes and k-Nearest Neighbor.

VIII. CONCLUSION

In this paper, the details of Artificial Neural Network and the various issues addressed in developing a classifier model using ANN is presented. A Feed Forward Neural Network trained using Back Propagation algorithm is developed for data classification. The performance of the developed ANN based classifier model has been tested using the Iris and Wine data set available in the UCI machine learning repository. For both the cases, simulation results show that the proposed neural classifier outperform the conventional classifiers.

But the developed neural classifier model has not performed well when applied to high dimensional data set. In the next paper, the importance of dimensionality reduction in ANN-based classifier model is discussed in detail

REFERENCES

[1] Denoeux T. (1995) 'A k-nearest neighbor classification rule based on Dempster-Shafer theory', IEEE Trans. Syst., Man, Cybern., Vol. 25, pp. 804–813

[2] Glorfeld L.W. (1996) 'A methodology for simplification and interpretation of back propagation-based neural networks models', Expert Syst. Application, Vol.10, pp.37-54.

[3] Guoqiang Peter Zhang (2000) 'Neural Networks for data classification: A Survey', IEEE Transaction on System Man and Cybernetics-Part C: Application and Reviews, Vol.30, No.4, pp.451-462.

[4] Haykin S. and Deng C. (1991) 'Classification of radar clutter using neural networks', IEEE Trans. on Neural Networks, Vol.2, No. 6, pp.589-600.

[5] Hu M.Y., Hung M.S., Shanker M.S. and Chen H. (1996) 'Using neural networks to predict the performance of Sino-foreign joint ventures', Int. J. of Comp. Intell. and Organ., Vol.1, No.3, pp.134-143.

[6] Jiawei Han and Micheline Kamber (2001) 'Data Mining: Concepts and Techniques', Morgan Kaufmann.

[7] Jurisica I. and Glasgow J.I. (1997) 'Improving Performance of Case-Based Classification Using Context Based Relevance', *International Journal on Artificial Intelligence Tools*, Vol. 6, pp. 511-536.