Fast Training of Multilayer Perceptrons

Brijesh Verma, *Member of IEEE & IASTED*
School of Information Technology
Faculty of Engineering and Applied Science
Griffith University, Gold Coast Campus
Gold Coast, Queensland 4217, Australia
E-mail: B.Verma@eas.gu.edu.au
URL: http://eassun.eas.gu.edu.au/

Abstract- **Training a multilayer perceptron by an error backpropagation algorithm is slow and uncertain. This paper describes a new approach which is much faster and certain than error backpropagation. The proposed approach is based on combined iterative and direct solution methods. In this approach, we use an inverse transformation for linearization of nonlinear output activation functions, direct solution matrix methods for training the weights of the output layer; and gradient descent, delta rule and other proposed techniques for training the weights of the hidden layers. The approach has been implemented and tested on many problems. Experimental results, including training times and recognition accuracy, are given. Generally, the approach achieves accuracy as good as or better than perceptrons trained using error backpropagation, and the training process is much faster than the error backpropagation algorithm and also avoids local minima and paralysis.**

Keywords: Artificial Neural Networks, Multilayer Perceptrons, Least Squares Methods, Supervised Learning, BackPropagation Algorithm, Local Minima.

I. INTRODUCTION

The Multi-Layer Perceptron (MLP) is a nonparametric technique for performing a wide variety of estimation tasks. Error Back-Propagation (EBP) [1] is one of the most important and widely used algorithms for training multilayer perceptrons. EBP has been applied to a wide variety of real world applications [2, 3].

One of the major problems of the EBP algorithm is the long and uncertain training process. For complex problems it may require days or weeks to train the network, and it may not train at all. Long training time can be the result of nonoptimum step size. Outright training failures generally arise from two sources: network paralysis and local minima [1, 2, 3]. A number of modifications [2, 3, 4, 5, 6, 7, 8, 9] have been proposed to improve the training of the EBP algorithm. Also many new training algorithms [10, 11, 12, 13, 14, 15, 16] for MLPs have been invented, but still we need a training method which could be faster and gives certainty for training MLPs.

In this paper we propose some training methods which are not only faster but also provide guaranteed training of MLPs. In the proposed methods, we use a transformation of nonlinear output activation function into linear functions and define a linear system of equations for the output layer. The linear system of equations is solved for calculating the weights of the output layer, using the modified Gram-Schmidt algorithm [17, 18]. The hidden layers are trained using EBP, delta rule and the other proposed techniques.

The major benefit of the new approach presented in this paper is its ability to train MLPs much faster than the conventional error backpropagation algorithm and results are comparable.

The remainder of this paper is organised as follows. In Section II we briefly present the notation for MLP description. In Section III we propose techniques for determining the weights of hidden layers and the output layer. In Section IV we present a description of the proposed methods in detail. In Section V computer simulations for performance evaluation and comparison are presented. Finally, Section VI concludes the paper.

II. NOTATION

Let n be the number of neurons in the input layer, m the number of neurons in the output layer, N_l be the number of neurons belonging to the *l*th layer and $o_k^{(l)}$ be the output of the *k*th neuron of the *l*th layer, then the computation performed by each neuron can be expressed as:

$$net_k^{(l)} = \sum_{j=1}^{N_{l-1}} w_{kj}^{(l)} o_j^{(l-1)} \tag{1}$$

$$o_k^{(l)} = f(\, net_k^{(l)} \,) \tag{2}$$

where $net_k^{(l)}$ is the weighted sum of the k neurons of the *l*th layer, $w_{kj}^{(l)}$ is the weight by which the same neuron multiplies the output $o_j^{(l-1)}$ of the *j*th neuron of the previous layer and f(.) is a nonlinear bounded function, often the sigmoid function.

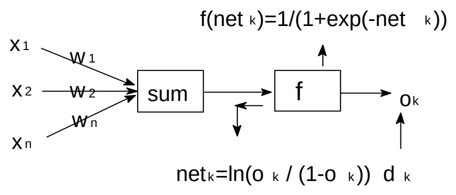

Figure 1. Single neuron model.

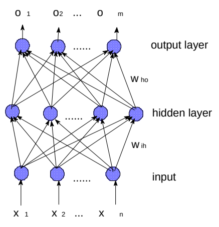

Figure 2. Multilayer Perceptron with single hidden layer.

Figure 1 shows a single neuron model. In the output layer for each neuron, the value *net* can be calculated as shown in Figure 1. We use this value to establish the linear system of equations for the output layer, which is described in Section III. Figure 2 shows a multilayer perceptron with single hidden layer suitable for training with the proposed approach. As shown in Figure 2, the input signal is simply passed through to the weights on their outputs. Each neuron in subsequent layers produces *net* and o_k signals as shown in Equations 1 and 2.

III. PROPOSED TECHNIQUES

In this Section we discuss Gradient Descent [1, 2], Delta Rule [2, 3], Radial Basis Function [2, 12] as well as some new techniques for training the weights of the hidden layer and a totally new technique is proposed for training the weights of the output layer.

A. Training the weights of the hidden layer.

The technique of unsupervised learning is often used to perform clustering as the unsupervised classification of objects without providing information about the actual classes. In our approach this kind of learning is used to adjust the weights of the hidden layer of MLP. For this purpose we propose to apply the following techniques:

1. Delta Rule (DR)

The delta rule [2,3] is only valid for continuous activation functions and in supervised training mode. Our aim in presenting the delta rule here, is to show that how this rule can be adapted in the proposed methods for training the weights of the hidden layer because this layer doesn't have target vectors.

Let $\underline{x}_1$, $\underline{x}_2$, ... $\underline{x}_p$ be the input vectors, and p be the total number of training pairs. The mapping of the input space into hidden units is shown in [12]. The input space consists of the set of all input vectors and hidden unit space consists of the set of the hidden units. We represent p-input vectors into (ln p / ln 2) bits (hidden units) in the hidden layer. Because p can be represented [12] by (ln p/ ln 2) bits.

In the proposed methods binary value 1/0 is defined as 0.9/0.1. Sometimes, we need more hidden units (> ln p / ln 2) to adjust more accurate weights of the output layer which would give more accurate results. The extra hidden units are adjusted to 0.1. After coding the hidden units, we apply the delta rule for training the hidden layer and weights can be adjusted as follows:

$$w_{ij}(n+1) = w_{ij}(n) + \Delta w_{ij}(n) \qquad (3)$$

$$\Delta w_{ij}(n) = \eta\ o\ (1-o)o_{pj}+\alpha(w_{ij}(n)-w_{ij}(n-1)) \qquad (4)$$

where

η = learning rate, α= momentum, w_{ij}= weight connecting nodes i and j, Δw_{ij}= change in weight value and o_{pj}= output of the node j at hidden layer.

2. Random Weights (RW)

The weights are adjusted at small random values. It is assumed that they fulfil the following conditions [12].

- The weights must be small real values (-0.5 +0.5) except zero. Small weights are recommended for good generalisation ability. The weights cannot be zero because if the weights are zero then we will get the same input vector for the output layer and the possibility of finding the weights of the output layer will be minimal.
- Two different input vectors cannot have the same output values for the hidden units. If for different input vectors, we have the same output values in the hidden layer and the desired values (targets) are different, then it is difficult to calculate the weights of the output layer.

3. Minimum Bit Distance (MBD)

The basic idea of MBD is to compute the measures of similarity in vector space for classification of the input vectors using unsupervised learning. The weights of the *i*th neuron of the hidden layer are input vectors of the *i*th pair. The minimum bit distance between vectors $\underline{x}$ and $\underline{w}$ is given by

$$MBD(\underline{x},\underline{w}) = ||\underline{x} - \underline{w}|| = \sqrt{(} \sum_{i=1}^{n} (x_i - w_i)^2 \ \frac{i}{n})$$

net = MBD($\underline{x}$,$\underline{w}$) (5b)

In Equation (5a), i/n is used to avoid the situation such as different vectors but same difference.

eg., let x be the [1,1,1], w1 be the [1,1,0] and w2 be the [0,1,1]. The MBD(x,w1) & MBD(x,w2) without i/n are the same. But as we can see they are not the same. In Equation (5a), factor i/n avoids this situation.

Output of the hidden layer is as follows:

o_k = (1 - tanh (net_k)) or, o_k = exp(-net_k) (6)

Note that different activation functions are used, which gives maximum value (o_k=1) at the output for best match (net_k = 0) [12].

B. Training the weights of the output layer

In supervised learning we assume that at each instance of time when the input is applied, the desired response d of the system is known. An important fact used in implementing a MLP is the relation between the desired response d and net. This relation can be found using Equation (2) and it can be provided as follows:

o_k=f(net_k)

o_k=1/(1+exp(-net_k)) (7)

or, net_k=ln(o_k/(1-o_k)) (8)

Because the desired response d_k is available for each neuron in the output layer and this value is equal to the o_k for output layer. It is easy to calculate the net from (8) by replacing the desired value d_k with the o_k value.

net_k=ln(d_k/(1-d_k)) (9)

where $0 < d_k < 1$

The weights of the output layer play an important role in multilayer perceptrons because they are directly connected with the output. The weights are easily trained using direct solution methods for linear equations. It is easy to calculate net value using the following equation

net_{ij} = ln ((d_{ij})/(1-d_{ij})) (10)

where net_{ij} = the value of net for the ith neuron and jth pair in the output layer.

d_{ij} = desired value dij of the ith neuron and jth pair in the output layer.

Using (2), Equation (10) becomes

$$w1i * x_{1j}^{l} + w2i * x_{2j}^{l} + .. + whi * x_{hj}^{l} = net_{ij}^{l} \quad (11)$$

where

w_{hi}=the value of a weight from neuron h in the hidden layer to neuron i in the output layer.

x_{hj}^{l}=the value of output for neuron h of pair j in the hidden layer l.

Using equation (11) given above, for each neuron in the output layer we can write a linear system

$$X\underline{w}=\underline{net} \quad (12)$$

where

$\underline{net}$ is p x 1, $\underline{w}$ is h x 1, p>=h and X is p x h with rank h.

In real world problems, p is usually greater than h; that is, there are more training patterns than hidden units. Therefore, we need to solve an overdetermined system of equations. In this study, many techniques have been tested and we have found that the Modified Gram-Schmidt (MGS) algorithm is very stable and needs less computer memory than other existing algorithms [17, 18]. In our approach, we use MGS to solve linear systems of equations (12).

IV. DESCRIPTION OF THE PROPOSED METHODS

1. Error Back-Propagation Using Direct Solutions: Method EBUDS

The basic idea of this method is to combine iterative and direct methods to improve the training time and avoid local minima and paralysis. This method can be applied to MLPs with any number of layers; however only two layers are needed to demonstrate the method and solve any real world problem.

This method uses a gradient search technique and direct solution methods (least square techniques) to minimise the cost function equal to the mean square difference between the desired and the actual net outputs. The desired output of all nodes is typically low (0.1 not 0) unless that node corresponds to the class of the current input, in which case it is high (0.9 not 1). The multilayer perceptron is trained by initially selecting small random weights and internal thresholds and then presenting all training data repeatedly. Weights are adjusted after each iteration and after some iteration or even when the training process is trapped at a local minimum; training of the MLPs can be stopped and the weights of the output layer can be calculated using direct solution methods. The method EBUDS can be outlined by the following steps:

1. Consider a two layer perceptron with a single hidden layer as shown in Figure 2.

2. Present input vectors ($\underline{x}_1, \underline{x}_2,...,\underline{x}_n$) and desired output vectors ($\underline{d}_1, \underline{d}_2,...,\underline{d}_n$).

3. Initialise all weights of the two layer perceptron at small random values.

   Use the RW technique from Section III.

4. Calculate actual outputs of the MLP.   Use Equations 1 and 2.

5. Train the weights of the hidden layer to output.   Use an error backpropagation technique for this purpose.

6. After a certain number of iterations stop the iterative training process.

7. Develop a linear system of equations for the output layer.

   Use Equation (10) and convert the output nonlinear activation function into linear function.

   Use Equation (11) and develop a linear system of equations as shown in Equation (12).

8. Calculate the weights of the output layer. Use the modified Gram-Schmidt algorithm to solve (12).

9. Repeat Step 7 through 8 for each neuron in the hidden layer.

2. Delta Rule-Symmetric Gaussian Elimination: Method DRSGE

In this method, the delta learning rule described in Section III is used for unsupervised (coded supervised) learning and the modified Gram-Schmidt algorithm or symmetric Gaussian elimination algorithm [17, 18] is used for supervised learning.

The delta rule is used to adjust the weights of the hidden layer. This layer is used to classify input data. For this classification we used two techniques, binary coding described in the previous section and the symmetric matrix coding which can be used in rare cases where the number of the hidden units is equal to the number of training pairs. In this technique, each input pair has a neuron in the hidden layer which has a desired value 0.9 and all other neurons in the hidden layer for this pair have the desired value of 0.1. The method DRSGE can be outlined by the following steps:

1. Steps 1, 2, 3 and 4 are the same as those of EBUDS.

2. Train the weights of the hidden layer.

   Use the delta learning rule which is described in Section III.

   Before using this rule, code the hidden units as shown above and in Section III.

   In the proposed methods, it is not necessary to perform full training of the weights of the hidden layer. Training the weights of the hidden layer can be stopped when suitable clusters are formed at the hidden units.

3. Steps 6, 7, 8 and 9 are same as that of EBUDS.

3. Random-Minimum Bit Distance-Gram-Schmidt: Method RMGS

The main idea of this method is to train an output layer by supervised learning and the remainder of the two layers by unsupervised learning. In this method the weights of the output layer are adjusted using modified Gram-Schmidt algorithm; the weights of the second hidden layer (which is directly connected to the output layer) are calculated using minimum bit distance (MBD) and the weights of the first hidden layer (which is directly connected to the input layer) are adjusted to small random values (RW).

The distance is treated as a measure of vector similarity. The minimum bit distance between vectors $\underline{x}$ and $\underline{w}$ is given by $MBD(\underline{x},\underline{w})=\|\underline{x}-\underline{w}\|$ [see Section III] (13)

where, $\underline{x}$ is an input vector and $\underline{w}$ is a weight vector of the second hidden layer.

In this approach, weights of the second hidden layer are input vectors of the first hidden layer.

If the number of training pairs is greater than the number of hidden units then the weights of the second hidden layer are randomly selected from training pairs as in Radial Basis Function type networks [12]. The method RMGS can be outlined by the following steps:

1. Consider a three layer perceptron with two hidden layers.
2. Initialise the weights of the first hidden layer to small random values. Use the RW technique from Section III.
3. Present input vectors ($\underline{x}_1, \underline{x}_2,...,\underline{x}_n$) and desired output vectors ($\underline{d}_1, \underline{d}_2,...,\underline{d}_n$).
4. Adjust the weights of the second hidden layer using MBD techniques from Section III.
5. Calculate actual outputs at the second hidden layer. Use Equations (1), (2), (5) and (6).
6. Develop a linear system of equations for the output layer.

   Use Equation (10) and convert the output nonlinear activation function into a linear function.

   Use Equations (11) and develop a linear system of equations as shown in Equation (12).
7. Calculate the weights of the output layer. Use the modified Gram-Schmidt algorithm to solve (12).
8. Repeat Step 6 through 7 for each neuron in the hidden layer.

4. The Proposed Methods (Comparisons and Problems)

The proposed methods are much faster and without local minima because they use direct solution methods. It means that they calculate weights of the output layer by converting nonlinear output functions into linear functions, and use a modified Gram-Schmidt method instead of iterative methods to find the weights of the output layer. A comparison with EBP is shown in the following section. We selected EBP because it is one of the most popular algorithms and implementation is very easy. It is very difficult to compare other existing methods because researchers use different environments for their experiments, implementation is not very easy and it is also time consuming. An implementation of the proposed methods is much more difficult than EBP because of the complicated MGS method.

V. EXPERIMENTAL RESULTS

The proposed methods have been tested on many problems such as the parity problem, exclusive nor problem, function approximation problem, sonar classification problem, vowel classification problem, real problems arising in pattern recognition, and on a variety of other problems from various sources. The methods have been implemented in C++ on an IBM PC and in C on a HP-UX 715/50 workstation.

As follows from literature, comparisons of training methods often use the number of iterations or epochs as a machine-independent measure of training time, but this is appropriate only when the methods being compared involve similar amounts of work per epoch. This is not the case when comparing the proposed methods with EBP. Thus the training times given below are the CPU times required for training on a HP-UX 715/50. In experiments, an iteration is said to be completed when all training patterns are presented and weights of the MLP are modified. The Root-Mean-Square (RMS) error is calculated using the formula [11,12].

Experiment 1 Approximation of a Function.

The task of training a function is a stringent one. In this task, a network is presented with some sampled points of a curve and it is asked to learn the training pairs and then generate an estimate of the original function.

In this experiment an application of the proposed approach to build a network which approximates the following function is presented:

$g(x)=0.2+0.8(x+0.7\sin(2\pi x))$

We assume $0 < x <= 1$. The training data are taken at intervals of 0.05; thus we have 21 data points:

$\{(0,g(0)), (0.05, g(0.05)),....,(1,g(1))\}$

We use 101 evaluation points taken at intervals of 0.01. The evaluation data are used to verify the interpolative power of the network.

The conventional error backpropagation did not reach the correct solution after 6000 iterations. The proposed methods took less than 200 iterations and 20 seconds. Some proposed methods which use only direct solution methods took even less than 3 seconds. All proposed methods for this problem are much faster than EBP. The interpolative and generalisation power of the proposed methods is quite good. The RMGS gives the best results and the method DRSGE is slightly better than EBP. The error is minimum for RMGS method, whereas the error for conventional EBP is maximum at some points. Finally, we can say that the results are quite good using the proposed methods for approximation of a function.

Experiment 2 Pattern Recognition Problem.

The proposed methods have been tested using a pattern recognition problem, consisting of the recognition of character patterns encoded as 8 x 8 pixel matrices, according to the IBM PC VGA character set. 36 different characters (0..9,A..Z) had to be recognised, corresponding to ASCII codes between 32 and 95. The number of input-output pairs is thus 36, and for each pair the input is a vector of 64 binary values, corresponding to the pixel matrix representing a character. The output is a vector of 7 binary values, representing its coded ASCII value. Therefore, the MLPs used have 64 inputs and 7 outputs. We set all input-output pairs of values of the training set to 0.1 and 0.9 rather than to zero and one respectively, to improve convergence. Network architectures with different numbers of hidden units are used for this problem. The results for best numbers of hidden units are presented in Table 1. Figure 3 depicts learning profiles produced for this problem and indicates that the proposed learning methods yield much faster learning.

Table 1. Comparative results on the pattern recognition problem.

Training method	# of hidden units	# of iterations	Training time [s]	Gain term η	Training set [%]	Test set [%]
EBP	20	800	300	0.5	100.00	97.20
EBUDS	20	100	45	0.5	100.00	97.20
RMGS	20	1	1.5	0.5	100.00	97.20
DRSGE	20	50	10	0.5	100.00	97.20

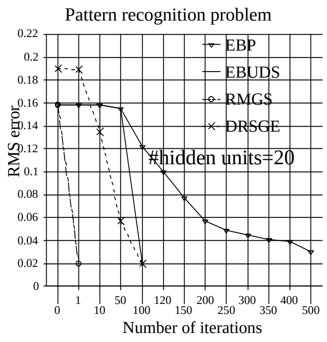

Figure 3. RMS vs. # of iterations for the pattern recognition problem.

Experiment 3 Sonar Problem, Mines vs. Rocks.

This is the data set used by Gorman and Sejnowski [19] in their study of the classification of sonar signals using a neural network. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The data set, is in the standard CMU Neural Network Benchmark format. The data has 60 continuous inputs and 1 enumerated output. This data set can be used in a number of different ways to test learning speed, quality of ultimate learning, ability to generalise, or combinations of these factors. There are 208 patterns in total with 111 belonging to the "metal" class and 97 belonging to the "rock" class. These 208 patterns are divided between the 104-member training set and the 104-member test set.

The purpose of this experiment is to compare the performances of the proposed methods and the conventional EBP algorithm with different numbers of hidden units. A learning rate of 0.2 and momentum of 0.5 is used. Errors less than 0.2 are treated as zero. Initial weights are uniform random values in the range -0.3 to +0.3.

Experimental Results showed that the performance of both the proposed methods and the conventional EBP algorithm goes up with the increase in the number of hidden units, and drops after a peak has been reached. The accuracy of all methods except RMGS goes up with the increase of the number of iterations till 350, and drops after 350 iterations for testing examples and remains unchanged for training examples (100%). Note that the proposed DRSGE method gives best performance (91%) for test examples and (100%) for training examples (Table 2), the other proposed method; EBUDS is second in performance and the last one is conventional error backpropagation. This experiment proves that the combined supervised and unsupervised training methods (DRSGE) can have better generalisation than the supervised EBP method.

Table 2. Comparative results on the sonar problem, Mines vs. Rocks

Training method	# of hidden units	# of iterations	Training time [s]	Gain term η	Training set % right on	Test set % right on
RMGS	88	1	15	0.0	100.00	83.00
EBP	22	500	300	0.5	100.00	86.50
EBUDS	22	150	65	0.5	100.00	86.50
DRSGE	22	60	100	0.5	100.00	86.5
	104	350	600	0.5	100.00	91.35

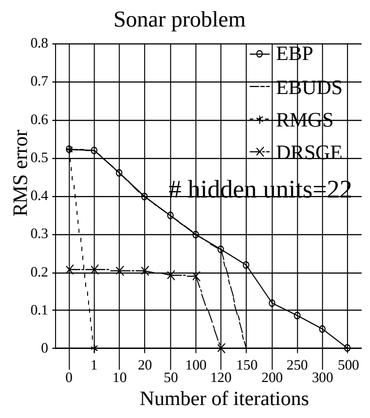

Figure 4. RMS vs. # of iterations for the sonar problem.

Experiment 4 Vowel Recognition Problem.

This is the data set used by Robinson [20] in their study of the speaker independent recognition of the eleven steady state vowels of British English using a specified training set of lpc derived log area ratios. The data set is in the standard CMU Neural Network Benchmark format. The data has 10 inputs and 11 outputs. Initial weights are uniform random values in the range -0.3 to +0.3. This data set can be used in a number of different ways to test learning speed, quality of ultimate learning, ability to generalise, or combinations of these factors. There are 860 patterns in total. These patterns are divided between the 430-member training set and the 430-member test set.

The networks used have 10 input nodes, 11 output nodes and have varying number of hidden units. A summary of results using different methods is presented in Table 3. As shown in Table 3 the training time for the proposed methods is less than the EBP method. The recognition rate for the proposed methods and the conventional EBP method is nearly the same, in some cases it is better for the proposed methods.

The experiment showed that the performance of both the proposed methods and conventional EBP for the training set goes up with the increase in the number of hidden units, and for the test set it goes up but drops after a peak has been reached. Figure 5 depicts the learning profiles produced for this problem and indicates that the proposed learning methods yield much faster learning.

Table 3. Comparative results on the vowel recognition problem.

Training method	# of hidden units	# of iterations	Training time [s]	Gain term η	Training set % right on	Test set % right on
RMGS	40	1	45	0.0	98.50	40.69
EBP	40	500	760	0.5	100.00	55.00
EBUDS	40	100	300	0.5	100.00	55.00
DRSGE	40	50	150	0.5	98.50	56.00

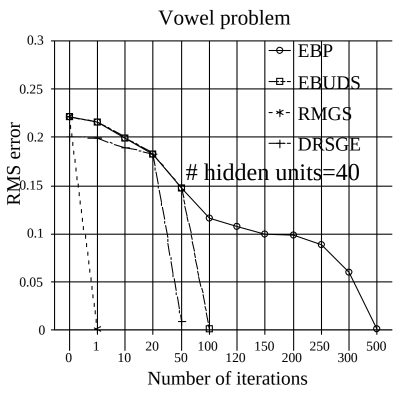

Figure 5. RMS vs. # of iterations for vowel recognition problem.

VI. CONCLUDING REMARKS

In this paper, we proposed methods for fast training of multilayer perceptrons. A number of experiments for classification as well as for approximation have been conducted and some of them are presented above. The experiments show that the proposed methods are without local minima and able to train the MLPs much faster than EBP. To demonstrate the generalisation and interpolative power of our methods we have presented some experimental results in Tables 1-3. Figures 3-5 show the comparison of RMS error vs. number of iterations for different problems. As follows from these tables and figures our methods performed well in all test cases considered and demonstrated a good generalisation capability. We conclude that our methods should be useful for many real world problems.

REFERENCES

[1]. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in Parallel Distributed Processing, vol. 1, Cambridge, MA: M.I.T. Press, 1986.

[2]. J. Hertz and A. Krogh, "Introduction to the Theory of Neural Computation," Addison-Wesley Publishing Company, CA, 1991.

[3]. P. D. Wasserman , "Neural Computing Theory and Practice," Van Nostrand Reinhold, NY, 1989.

[4]. J. J. Mulawka, and B. K. Verma, "Improving the Training Time of the Backpropagation Algorithm," International Journal of Microcomputer Applications, vol. 13, no.2, pp.85-89, Canada, 1994.

[5]. Y. Liping, and Y. Wanzhen, "Backpropagation with Homotopy," Neural Computation, vol. 5, no. 3, pp. 363-366, 1993.

[6]. B. K. Verma and J.J. Mulawka, "A Modified Backpropagation Algorithm," Proc. of the World Congress on Computational Intelligence, vol.2, pp.840-846 26 June-2 July, Orlando, USA, 1994.

[7]. Y. Yam and T. Chow, "Extended Backpropagation Algorithm," Electronic Letters, vol. 29, no. 19, pp.1701-1702, 1993.

[8]. J. A. Freeman, "BP and its variants," Addison-Wesley Publishing Company, Inc., NY. 1994.

[9]. M. Fukumi and Sigeru Omatu, "A New Back-Propagation Algorithm with Coupled Neuron," IEEE Trans. on Neural Networks, vol. 2, no. 5, pp.535-538.

[10]. F. Biegler and F. Barmann, "A Learning Algorithm for Multilayer Perceptrons:Layer by Layer," IEEE Trans. on Neural Networks," vol. 6, no. 1, 1995.

[11]. B. K. Verma and J. J. Mulawka, "Training of the Multilayer Perceptron Using Direct Solution Methods," Proc. of the Twelfth IASTED International Conference, Applied Informatics, pp. 18-24, May 17-20, Annecy, France, 1994.

[12]. B. K. Verma, "New methods of Training the MLP," PhD Dissertation, Warsaw Univ. of Technology, Warsaw, 1995.

[13]. S. D Hunt and J.R Deller, "Efficient Training of Feedforward Artificial Neural Networks Based on Matrix Perturbation Theory," Submitted to International Journal on Neural Systems, London, UK.

[14]. Andre Neubauer, "Robust Learning Algorithms for Multi-Layer Perceptrons with Discretized Synaptic Weights," Proc. of the IEEE International Conference on Neural Networks, ICNN'95, pp. 3154, Australia, 1995.

[15]. M. H Brugge and J.A. Nijhuis, "On the Representation of Data for Optimal Learning," Proc. of the IEEE International Conference on Neural Networks, ICNN'95, pp. 3180, Australia, 1995.

[16]. M. R. Azmi and R. Liou, "Fast learning process of MLP NNs using Recursive Least Squares Methods," IEEE Trans. on Signal Processing, vol. 4, no. 2, pp. 446-450, 1993.

[17]. A. Kielbasinski and H. Schwetlick, "Numerical Linear Algebra," Warsaw, 1992.

[18]. A. Bjorck and G. Dahlquist, G., "Numerical Methods," Printice-Hall, Inc., New Jersey, 1974.

[19]. R. P. Gorman and T. J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," Neural Networks, vol. 1, pp. 75-89, 1988.

[20]. A. J. Robinson and F. Fallside, "Static and Dynamic Error Propagation Networks with Applications to Speech Coding," Neural Information Processing Systems, pp. 632-641, USA, 1988.