A VARIANT OF BACK-PROPAGATION ALGORITHM FOR MULTILAYER FEED-FORWARD NETWORK

Anil Ahlawat, Sujata Pandey

Abstract: In this paper, a variant of Backpropagation algorithm is proposed for feed-forward neural networks learning. The proposed algorithm improve the backpropagation training in terms of quick convergence of the solution depending on the slope of the error graph and increase the speed of convergence of the system. Simulations are conducted to compare and evaluate the convergence behavior and the speed factor of the proposed algorithm with existing Backpropagation algorithm. Simulation results of large-scale classical neural-network benchmarks are presented which reveal the power of the proposed algorithm to obtain actual solutions.

Keywords: Backpropagation, convergence, feed-forward neural networks, training.

Introduction

Feed-forward neural networks (FNN) have been widely used for various tasks, such as pattern recognition, function approximation, dynamical modeling, data mining, and time series forecasting, [1-3]. The training of FNN is mainly undertaken using the back-propagation (BP) based learning. The back-propagation algorithm has been investigated many times with minor variations [4-7]. However, even to date, there are still a great number of problems that cannot be solved efficiently by the majority of the training algorithms that have been proposed over the years, using standard simple feed-forward network architectures. A number of different kind of BP based learning algorithms, such as an on-line neural-network learning algorithm for dealing with time varying inputs [8], fast learning algorithms based on gradient descent of neuron space [9], second derivative based non-linear optimization methods [10], conjugate gradient methods [11] and genetic algorithms [12,13] avoid use of any gradient information. Levenberg–Marquardt algorithm [14-16] is the most powerful and a popular second derivative based algorithm that have been proposed for the training of feed-forward networks which combines the excellent local convergence properties of Gauss-Newton method near a minimum with the consistent error decrease provided by (a suitably scaled) gradient descent faraway from the solution. In first-order methods (such as gradient descent), a local minimizer problem is overshooted with the inclusion of momentum term. The momentum term actually inserts second-order information in the training process and provides iterations whose form is similar to the conjugate gradient (CG) method. The major difference of Backpropagation with the conjugate gradient method is that the coefficients regulating the weighting between the gradient and the momentum term are heuristically selected in BP, whereas in the CG algorithm these coefficients are adaptively determined. However, these algorithms also share problems [17] present in the standard Backpropagation algorithm and may converge faster in some cases and slower in others. Comparison of the speeds of convergence of different schemes for implementing Backpropagation is not clear-cut, though a discussion on benchmarking of the algorithms can be found [18].

In this paper, a proposal for a variant of back-propagation algorithm for FNN with time-varying inputs has been presented which is capable of overcoming the shortcomings of the BP as discussed above. In the experimental section, the proposed algorithm is compared with the existing Backpropagation algorithm for training multilayer feed-forward networks on training tasks that are well known for their complexity. It was observed that the proposed algorithm have shown to solve these tasks with exceptionally high success rates and converged much faster than the original BP algorithm and showed greater accuracy.

Backpropagation Algorithm

Overview of the Algorithm

The Backpropagation training algorithm [1] for training feed-forward networks was developed by Paul Werbos [7], and later by Parker [4] and Rummelhart [5]. This type of network configuration is the most common in use, due to its ease of training [19]. It is estimated that over 80% of all neural network projects in development use back-propagation. In back-propagation, there are two phases in its learning cycle, one to propagate the input pattern through the network and the other to adapt the output, by changing the weights in the network. It is the error signals that are back propagated in the network operation to the hidden layer(s). The portion of the error signal that a hidden-layer neuron receives in this process is an estimate of the contribution of a particular neuron to the output error. Adjusting on this basis the weights of the connections, the squared error, or some other metric, is reduced in each cycle and finally minimized, if possible.

A Back-Propagation network consists of at least three layers of units: an input layer, at least one intermediate hidden layer, and an output layer. Typically, units are connected in a feed-forward fashion with input units fully connected to units in the hidden layer and hidden units fully connected to units in the output layer. When a Back-Propagation network is cycled, an input pattern is propagated forward to the output units through the intervening input-to-hidden and hidden-to-output weights.

Training in Backpropagation Algorithm

The feed-forward back-propagation network undergoes supervised training [20], with a finite number of pattern pairs consisting of an input pattern and a desired or target output pattern. An input pattern is presented at the input layer. The neurons here pass the pattern activations to the next layer neurons, which are in a hidden layer. The outputs of the hidden layer neurons are obtained by using a bias, and also a threshold function with the activations determined by the weights and the inputs. These hidden layer outputs become inputs to the output neurons, which process the inputs using an optional bias and a threshold function. The final output of the network is determined by the activations from the output layer.

The computed pattern and the input pattern are compared, a function of this error for each component of the pattern is determined, and adjustment to weights of connections between the hidden layer and the output layer is computed. A similar computation, still based on the error in the output, is made for the connection weights between the input and hidden layers. The procedure is repeated with each pattern pair assigned for training the network. Each pass through all the training patterns is called a cycle or an epoch. The process is then repeated as many cycles as needed until the error is within a prescribed tolerance. The adjustment for the threshold value of a neuron in the output layer is obtained by multiplying the calculated error in the output at the output neuron and the learning rate parameter used in the adjustment calculation for weights at this layer.

After a Back-Propagation network has learned the correct classification for a set of inputs from a training set, it can be tested on a second set of inputs to see how well it classifies untrained patterns. Thus, an important consideration in applying Back-Propagation learning is how well the network generalizes.

Mathematical Analysis of the Algorithm

Assume a network with N inputs and M outputs. Let x_i be the input to ith neuron in input layer, B_j be the output of the jth neuron before activation, y_j be the output after activation, b_j be the bias between input and hidden layer, b_k be the bias between hidden and output layer, w_{ij} be the weight between the input and the hidden layers, and w_{jk}

be the weight between the hidden and output layers. Let ɳ be the learning rate and ƍ the error. Also, let i, j and k be the indexes of the input, hidden and output layers respectively.

The response of each unit is computed as:

$$B_j = b_j + \sum_{i=1}^{n} x_i \cdot w_{ij} \tag{1}$$

$$y_j = \left(1/\left(1+\exp(-B_j)\right)\right) \tag{2}$$

Weights and bias between input and hidden layer are updated as follows:

For input to hidden layer, for i = 1 to n,

$$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j y_i \tag{3}$$

$$b_j(t+1) = b_j(t) + \eta\delta_j \tag{4}$$

Where, δ_j is the error between the input and hidden layer and calculated as:

$$\delta_j = y_j \cdot (1 - y_j) \cdot \sum_k \delta_k w_{jk} \tag{5}$$

Weights and bias between hidden and output layer are updated as follows:

For hidden to output layer, for j = 1 to h,

$$w_{jk}(t+1) = w_{jk}(t) + \eta\delta_k y_j \tag{6}$$

$$b_k(t+1) = b_k(t) + \eta\delta_k \tag{7}$$

and δ_k is the error between the hidden and output layer and calculated as:

$$\delta_k = y_k \cdot (1 - y_k) \cdot (d_k - y_k) \tag{8}$$

Proposed variant of Backpropagation Algorithm

The Backpropagation algorithm described above has many shortcomings [17]. The time complexity of the algorithm is high and it gets trapped frequently in sub-optimal solutions. It is also difficult to get an optimum step size for the learning process, since a large step size would mean faster learning, which may miss an optimal solution altogether, and a small step size would mean a very high time complexity for the learning process. The proposed variant of the Backpropagation algorithm aims to overcome some of these shortcomings.

Overview of the Proposed Algorithm

The Backpropagation Algorithm described above is modified by following changes:

1 Momentum: A simple change to the training law that sometimes results in much faster training is the addition of a momentum term [21]. With this change, the weight change continues in the direction it was heading. This weight change, in the absence of error, would be a constant multiple of the previous weight change. The momentum term is an attempt to try to keep the weight change process moving, and thereby not gets stuck in local minima's. In some cases, it also makes the convergence faster and the training more stable.

2 Dynamic control for the learning rate and the momentum: Learning parameters such as Learning rate and momentum serve a better purpose if they can be changed dynamically during the course of the training [21]. The learning rate can be high when the system is far from the goal, and can be decreased when the system gets nearer to the goal, so that the optimal solution cannot be missed.

3 Gradient Following: Gradient Following has been added to enable quick convergence of the solution depending on the slope of the error graph. When the system is far away from the solution, the learning rate is further increased by a constant parameter C1 and when the system is close to a solution, the learning rate is decreased by a constant parameter C2. The farness or closeness of the system from the solution was determined from the slope of the Error graph [22-26].

4 Speed Factor: To increase the speed of convergence of the system, a speed factor S has been used as determined by a mathematical formula derived from the study of graphs.

Mathematical Analysis of the Algorithm

1. Momentum: Let the momentum term be α. Then equation (3) and equation (4) would be modified as:

For input to hidden layer, for i = 1 to n,

$$w_{ij}(t+1) = w_{ij}(t) + \eta\delta_j y_i + \alpha \cdot (w_{ij}(t) - w_{ij}(t-1)) \tag{9}$$

$$b_j(t+1) = b_j(t) + \eta\delta_j + \alpha \cdot (b_j(t) - b_j(t-1)) \tag{10}$$

The term δ_j would be calculated as in equation (5).

The equation (6) and equation (7) would be modified as:

For hidden to output layer, for j = 1 to h,

$$w_{jk}(t+1) = w_{jk}(t) + \eta\delta_k y_j + \alpha \cdot (w_{jk}(t) - w_{jk}(t-1)) \tag{11}$$

$$b_k(t+1) = b_k(t) + \eta\delta_k + \alpha \cdot (b_k(t) - b_k(t-1)) \tag{12}$$

The term δ_k would be calculated as in equation (8).

2. Dynamic Control for learning rate and momentum: If changing the weight decreases the cost function (mean squared error), then the learning rate is given by

$$\eta = \eta + 0.05 \tag{13}$$

Else

$$\eta = \eta - 0.05 \tag{14}$$

Similar conditions were placed for the momentum term α.

3. Gradient Following: Let C1 and C2 be two constants, such that C1 > 1 and 0 < C2 < 1 and Δmax and Δmin be the maximum and minimum change permissible for the weight change. If $\frac{\partial E}{\partial w}$ is the gradient following term then three cases need to be considered:

a) Case I

If $\frac{\partial E}{\partial w}(t) \cdot \frac{\partial E}{\partial w}(t-1) > 0$

Then

$$\Delta(t) = \min(\Delta(t) \,.\, C1, \Delta\max) \tag{15}$$

$$\Delta w = -\frac{\partial E}{\partial w}(t) \cdot \Delta(t) \tag{16}$$

$$w(t+1) = w(t) + \Delta w \tag{17}$$

b) Case II

If $\frac{\partial E}{\partial w}(t) \cdot \frac{\partial E}{\partial w}(t-1) < 0$

Then

Δ(t) = min(Δ(t) . C2, Δmin) (18)

$$\frac{\partial E}{\partial w}(t) = 0 \quad (19)$$

w(t+1) = w(t) - Δw (20)

c) Case III

If $\frac{\partial E}{\partial w}(t) \cdot \frac{\partial E}{\partial w}(t-1) = 0$

Then

$$\Delta w = -\frac{\partial E}{\partial w}(t) \cdot \Delta(t) \quad (21)$$

w(t+1) = w(t) + Δw (22)

4. Speed Factor: Let S be the speed factor. Then equation (9) and equation (10) would further be modified to:

For input to hidden layer, for i = 1 to n,

$$w_{ij}(t+1) = w_{ij}(t) + S\left[\eta\delta_j y_i + S \cdot \alpha \cdot (w_{ij}(t) - w_{ij}(t-1))\right] \quad (23)$$

$$b_j(t+1) = b_j(t) + S\left[\eta\delta_j + S \cdot \alpha \cdot (b_j(t) - b_j(t-1))\right] \quad (24)$$

Similarly, equation (11) and (12) would be modified as:

For hidden to output layer, for j=1 to h,

$$w_{jk}(t+1) = w_{jk}(t) + S\left[\eta\delta_k y_j + S \cdot \alpha \cdot (w_{jk}(t) - w_{jk}(t-1))\right] \quad (25)$$

$$b_k(t+1) = b_k(t) + S\left[\eta\delta_k + S \cdot \alpha \cdot (b_k(t) - b_k(t-1))\right] \quad (26)$$

Experimental Study

The algorithm proposed in this paper were tested on the training of standard multilayer feed forward networks (FNNs) and applied to several problems. The FNN simulator was implemented in Visual Basic .NET. The performance of the proposed algorithm was compared to existing Backpropagation algorithm. All simulations were carried out on a Pentium IV 2 GHz with 128 MB RAM PC using the FNN simulator developed by our team.

Number of cycles	BP (time in msec)	Speed1 (time in msec) for momentum = 0.1 and speed = 0.1	Speed2 (time in msec) for momentum = 0.1 and speed = 0.2	Speed3 (time in msec) for momentum = 0.2 and speed = 0.1
100	42781.52	4626.66	4927.08	5668.15
300	123968.25	9333.43	9754.03	10094.51
500	206146.42	15442.20	15452.21	15552.36
800	330385.06	24204.80	25666.91	24585.36

1000	414546.10	30173.39	31054.64	30383.69
1200	496964.60	35671.28	36612.64	36712.79
1500	617187.47	44954.65	46096.28	46076.26

Table 1: Comparison of training time between Backpropagation algorithm
and proposed algorithm for different momentum
and speed for 8-bit parity problem.

The selection of initial weights is important in feed-forward neural network training. If the initial weights are very small, the backpropagated error is so small that practically no change takes place for some weights, and therefore more iteration are necessary to decrease the error. If the error remains constant, then the learning stops in an undesired local minimum. Large values of weights, results in speed up of learning, but they can lead to saturation and to flat regions of the error surface where training is slow. Keeping these in consideration, the experiments were conducted using the same initial weight vectors that have been randomly chosen from a uniform distribution in (-1,1). Sensitivity of the algorithm in some other intervals (-0.1,0.1) was also studied to investigate its convergence behavior.

Number of cycles	BP (time in msec)	Speed1 (time in msec) for momentum = 0.1 and speed = 0.1	Speed2 (time in msec) for momentum = 0.1 and speed = 0.2	Speed3 (time in msec) for momentum = 0.2 and speed = 0.1
100	2072.9856	1482.1376	1412.0320	1442.0736
300	4256.1152	2022.9120	1832.6400	1592.2944
500	6399.2064	2022.9120	2563.6864	2363.3920
800	9183.2064	2022.9120	3815.4880	3585.1648
1000	11156.0448	2022.9120	3945.6768	3895.6032
1200	13038.7584	2022.9120	4746.8288	5097.3184
1500	16714.0352	2022.9120	5808.3584	5588.0320

Table 2: Comparison of training time between Backpropagation algorithm and proposed algorithm for different momentum and speed for Hurst Motor.

The initial learning rate was kept constant for both algorithms. It was chosen carefully so that the Backpropagation training algorithm rapidly converges without oscillating toward a global minimum. Then all the other learning parameters were tuned by trying different values and comparing the number of successes exhibited by five simulation runs that started from the same initial weights.

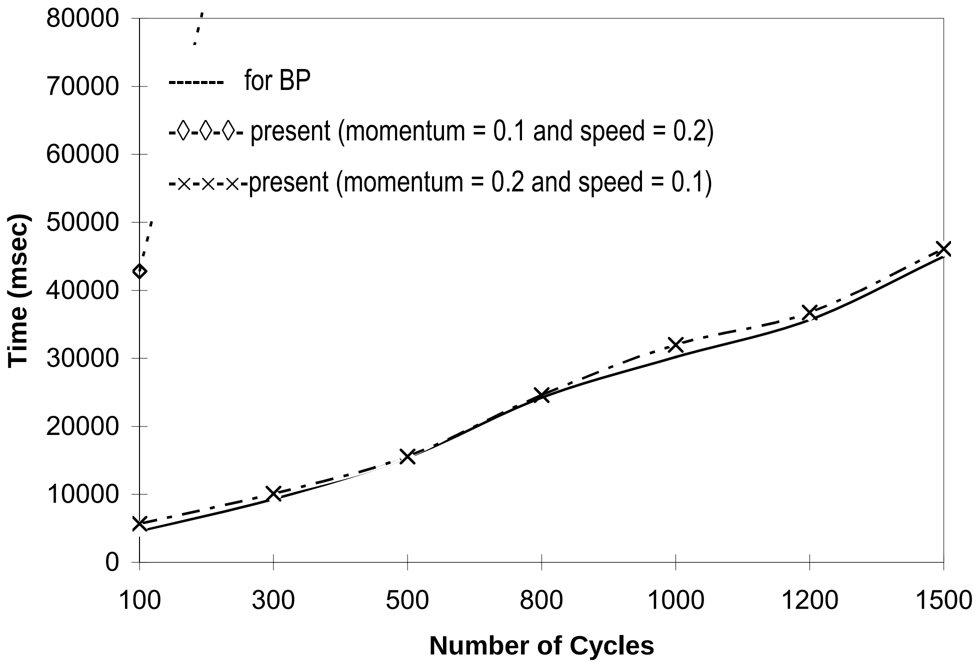

Fig. 1. Variation of time with number of cycles for 8-bit parity problem.

To obtain the best possible convergence, the momentum term and the speed constant are normally adjusted by trial and error or even by some kind of random search. Since the optimal value is highly dependent on the learning task, no general strategy has been developed to deal with this problem. Thus, the optimal value of these two terms is experimental but depends on the learning rate chosen. In our experiments, we have tried eight different values for the momentum ranging from 0.1 to 0.8 and for speed constant, we have tried five different values ranging from 0.1 to 0.5 and we have run five simulations combining all these values with the best available learning rate for BP. But it was shown that some combinations give better results, which is shown in Table 1 for 8-bit parity problem and in Table 2 for Hurst motor. On the other hand, it is well known that the "optimal" learning rate must be reduced when momentum is used. Thus, we also tested combinations with reduced learning rates.

Table 1 shows the results of training on 8-8-1 network (eight inputs, one hidden layer with eight nodes and one output node) on the 8-bit parity problem. It can be observed that training is considered successful for the given dataset for speed constant and momentum in table 1. It can be seen from the table 1 that training time is drastically reduced in the proposed algorithm. Figure 1 shows the variation of training time with number of cycles (epoch) for the Backpropagation and the three different cases for the proposed algorithm for 8-bit parity problem. In BP for increase in number of cycles, the training time increases rapidly but in all the cases for the proposed speed algorithm the training time increases gradually. Also for the change in the momentum and speed term, there was not much difference in the training time.

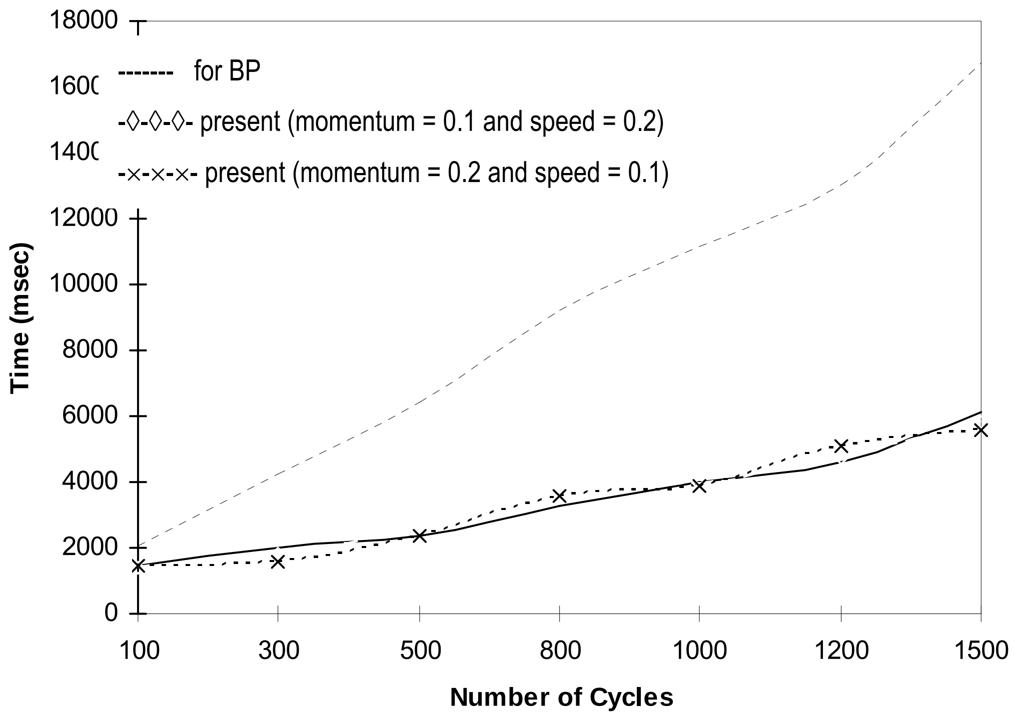

Fig. 2. Variation of time with number of cycles for Hurst Motor.

Table 2 shows the results of training on 5-8-2 network (five inputs, one hidden layer with eight nodes and two output node) for the Hurst motor. It was observed that training is also considered successful for the given dataset for speed constant and momentum in table 2. It can also be seen from the table 2 that training time is drastically reduced in the proposed algorithm. Figure 2 shows the variation of training time with number of cycles (epoch) for the Backpropagation and the three different cases for the proposed algorithm for Hurst motor. In BP for increase in number of cycles, the training time increases rapidly but in all the cases for the proposed speed algorithm the training time increases gradually. Also for the change in the momentum and speed term, there was not much difference in the training time.

Conclusion

The variant in BP has been proposed for the training of feed forward neural networks. The convergence properties of both algorithm have been studied and the conclusion was reached that new algorithm is globally convergent. The proposed algorithm was tested on available training tasks. These results point to the conclusion that the proposed methods stand as very promising new tools for the efficient training of neural networks in terms of time. It also proves to be much more accurate than the existing Backpropagation algorithm. In addition the error correction rate achieved is much faster and training time is also much faster as shown in the results.

The proposed variant has a lower slope signifying a faster training compared to the Backpropagation algorithm and it converges to a more stable solution thereby ending the learning process. The Backpropagation algorithm on the other hand, may not converge at all throughout the learning process.

Bibliography

1 J. M. Zurada, "Introduction to artificial neural systems," *M. G. Road, Mumbai: Jaico,* (2002).

2 P. Mehra and B. W. Wah, "Artificial neural networks: concepts and theory," *IEEE Comput. Society Press*, (1992).

3 Bob Waterbury, "Artificial intelligence expands frontiers in asset management, condition monitoring and predictive maintenance systems make AI pay off with lower maintenance costs, improved forecasting, and fewer unplanned shutdowns," *Putman Media*, (November 16, 2000).

4 D.B Parker, "Learning logic, technical report TR-47," *Center for Computational Research in Economics and Management Sciences*, MIT, (1985).

5 D.E. Rumelhart, G.E Hinton, and R.J. Williams, "Learning internal representations by error propagation," *Parallel Distribution Processing*, vol. 1, pp. 318-362. MIT Press, Cambridge, MA (1986).

6 P.J. Werbos, "The roots of backpropagation," *John Wiley and Sons, Inc.*, New York (1994).

7 T. Nitta, "An analysis on decision boundaries in the complex-backpropagation network," *IEEE World Congress on Computational Intelligence*, vol. 2, pp. 934-939, Orlando, FL, June 1994, IEEE Computer Society Press.

8 Y. Zhao, "On-line neural network learning algorithm with exponential convergence rate," *Electron. Lett.*, vol. 32, no. 15, pp. 1381–1382 (July 1996).

9 G. Zhou and J. Si, "Advanced neural network training algorithm with reduced complexity based on Jacobian deficiency," *IEEE Trans. Neural Networks*, vol. 9, pp. 448–453 (May 1998).

10 D.B. Parker, "Optimal algorithms for adaptive networks: second order backpropagation, second order direct propagation and second order Hebbian learning," *First IEEE International Conference on Neural Networks*, San Diego, pp. 593-600 (1987).

11 E.M. Johansson, F.U. Dowla, and D.M. Goodman, "Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method," *Intl. J. Neural Systems*, vol. 2: pp. 291-301 (1992).

12 D.J. Mortana and L. Davis, "Training feed-forward networks using genetic algorithms," *In Proceedings of 11th Intl. Joint Conf. in Artificial Intelligence (IJCAI)*, Detroit, MI, pp. 762-767, Morgan Kaufmann, San Mateo, CA (1989).

13 D. Whitley and T. Hanson, "Optimizing neural networks using faster, more accurate genetic search," *Proceedings of 3rd Intl. Conf. Genetic Algorithms*, pp 391-396, Morgan Kaufmann, San Mateo, CA (1989).

14 R. Parisi, E. D. Di Claudio, G. Orlandi, and B. D. Rao, "A generalized learning paradigm exploiting the structure of feed-forward neural networks," *IEEE Trans. Neural Networks*, vol. 7, pp. 1450–1459, Nov. (1996).

15 M. T. Hagan and M. B. Menhaj, "Training feed-forward neural networks with the Marquardt algorithm," *IEEE Trans. Neural Networks*, vol. 5, pp. 989–993, Nov. (1994).

16 X. Yu, M. O. Efee, and O. Kaynak, "A general backpropagation algorithm for feed-forward neural networks learning," *IEEE Trans. Neural Networks*, vol. 13, no. 1, pp. 251–254 (January 2002).

17 S. Saarinen, R. Brambley, and G. Cybenko, "Ill-conditioning in neural network training problems," *SAIM J. Sci. Comput*., 14(3):693-714 (May 1993).

18 S. E. Fahlman, "Faster-Learning variations on backpropagation: an empirical study in D. Touretzky, G. Hinton, and T. eds. Sejnowski," *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 38-51(1998).

19 J.S. Judd, "Neural Network Design and the complexity of Learning," *MIT Press*, Cambridge, MA (1990).

20 D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Processing Magzine*, vol. 10, no. 1, pp. 8-39 (1993).

21 R.A. Jacobs, "Increased rate of convergence through learning rate adaptation," *Neural Networks*, pp 295-307 (1988).

22 P.R. Adby and M.A.H. Dempster, "Introduction to Optimization Methods," *Haisted Press*, New York (1974).

23 L. Cooper and D. Steinberg, "Introduction to Methods of Optimization," *Saunders, Philadelphia* (1970).

24 R.W. Daniels, "An introduction to numerical methods and optimization techniques," *North-Holland*, New York, 1978.

25 Karl-Heinz Elster, editor, “Modern Mathematical Methods of Optimization,” *VCH*, New York (1993).

26 R. Fletcher, “Practical Methods of Optimization,” *2nd ed. Wiley*, New York (1987).

Authors' Information

Anil Ahlawat, Sujata Pandey – Department of Computer Science and Engineering, Amity School Of Engineering and Technology, 580, Delhi Palam Vihar Road, Bijwasan, New Delhi, 110061, India; e-mail: a_anil2000@yahoo.com