Scientia Iranica, Vol. 8, No. 4, pp 250–264
© Sharif University of Technology, October 2001

SCIENTIA
IRANICA

Adaptation of Momentum Factor and Steepness Parameter in Backpropagation Algorithm Using Fixed Structure Learning Automata

H. Beigy* and M.R. Meybodi[1]

Backpropagation (BP) algorithm is a systematic method for training multi-layer neural networks, which, despite many successful applications, also has many drawbacks. For complex problems, backpropagation may require a long time to train the networks and it is possible that no training occurs at all. Long training time can be the result of non-optimal parameters. It is not easy to choose an appropriate value for the parameters of a particular problem and the parameters are usually determined by trial and error. If the parameters are not chosen appropriately, slow convergence, paralysis and continuous instability can result [1-4]. Moreover, the best values for the parameters at the beginning of training may not be good enough later. In this paper, a technique has been incorporated into BP algorithm for adaptation of steepness parameter and momentum factor in order to achieve a higher rate of convergence. Through interconnection of Fixed Structure Learning Automata (FSLA) to the feedforward neural networks, learning automata scheme is applied in order to adjust these parameters based on the observation of random response of neural networks. The main motivation in using learning automata as an adaptation algorithm is in its capability of global optimization when dealing with multi-modal surfaces. The feasibility of the proposed method is shown through simulations on three learning problems: exclusive-or, encoding problem and digit recognition. These problems are chosen because they have different error surfaces and collectively present an environment that is suitable to determine the effect of the proposed method. The simulation results show that the adaptation of these parameters using this method increases not only the convergence rate of learning but also the likelihood of escaping the local minima. Computer simulations provided in this paper indicate that at least a magnitude of savings in running time can be achieved when FSLA is used for the adaptation of momentum factor and steepness parameters. Furthermore, simulations demonstrate that the FSLA approach performs much better than the Variable Structure Learning Automata (VSLA) approach reported in [1,2].

INTRODUCTION

Error backpropagation training algorithm (BP), an iterative gradient descent algorithm, is a simple way to train multi-layer feedforward neural networks [5]. The backpropagation algorithm is based on the gradient descent rule:

$$\Delta w_{jk}(n) = -\alpha \frac{\partial E}{\partial w_{jk}} + \mu \Delta w_{jk}(n-1), \tag{1}$$

*. *Corresponding Author, Department of Computer Engineering, Amir Kabir University of Technology, Tehran, I.R. Iran.*

1. *Department of Computer Engineering, Amir Kabir University of Technology, Tehran, I.R. Iran.*

where w_{jk} is the weight on the connection outgoing from unit j and entering unit k, α, μ and n are learning rate, momentum factor and time index, respectively. In the BP framework, α and μ are constant and E is defined as:

$$E(n) = \frac{1}{2} \sum_{p=1}^{\neq \text{patterns}} \sum_{j=1}^{\text{outputs}} (T_{p,j} - O_{p,j})^2, \tag{2}$$

where $T_{p,j}$ and $O_{p,j}$ are desired and actual outputs for pattern p at output node j and the index p varies on the training set. In the BP algorithm framework, each computational unit computes the same activation function. The computation of sensitivity for each neuron requires the derivative of activation function, therefore, this function must be continuous. The activation function is normally a sigmoid function chosen between the two following functions:

$$f(x)=\frac{1}{1+e^{-\lambda x}}, \tag{3}$$

$$f(x)=\frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}. \tag{4}$$

The steepness parameter λ determines the active region (region in which the derivative of sigmoid function is not very small) of the activation function. As the steepness parameter decreases from positive infinity to zero, (Figure 1) the sigmoid function changes from a unit step function to constant value of 0.5.

Figure 2 shows the derivative of sigmoid function in its active region where the derivative is greater than 0.01. For large values of the steepness parameter λ, the derivative is very large and the active region of sigmoid function is very small. In this region, the high value of the derivative forces the algorithm to oscillate. A small active region means that the weights are updated rarely. For small values of steepness parameter λ, the active region is very wide, but the derivative is

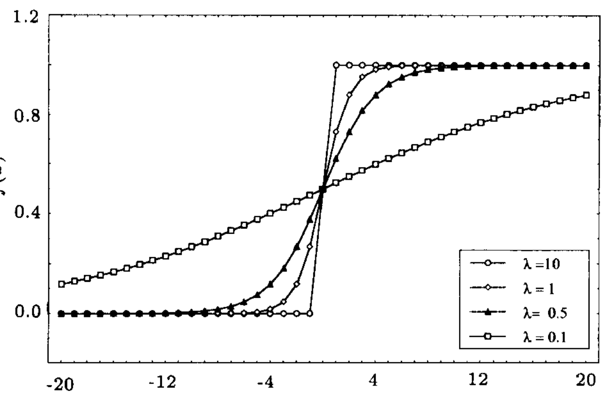


Figure 1. Sigmoid function with different steepness parameter λ.

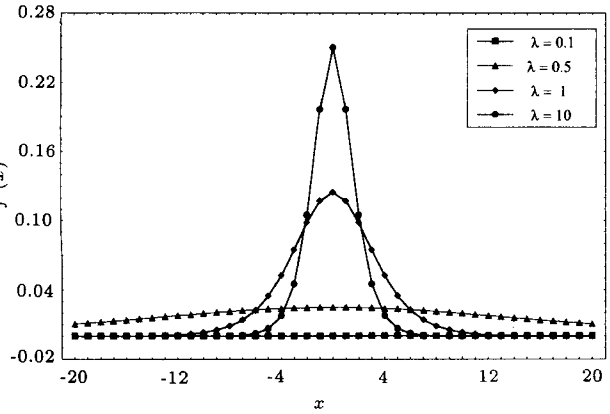


Figure 2. Derivative of sigmoid function in its active range.

very small and speed of convergence very low. The steepness parameter λ is often set to a constant value and is not changed by the learning algorithm. Much flexibility is gained if the net inputs of the sigmoidal functions are moved near to their active regions, where the associated gradients are not very close to zero. This enables the BP algorithm to avoid some points in the network parameters space where the BP algorithm would effectively stop, even though it is not close to a local minima point. This will cause the gradient of the error function to be small if the sigmoidal is shifted far outside the active region of the input to the function. Therefore, it is desirable to center each sigmoid to be inside the active region of the sigmoidal function.

The momentum term in Equation 1 causes great changes in weights ($\Delta w_{jk}(n)$), if the current changes of weights ($\Delta w_{jk}(n-1)$) are large and causes small changes in weights if the current changes are small. This means that the network is less likely to get stuck in local minima early on, since the momentum term pushes the changes towards a local downward trend.

Momentum is of great assistance in speeding up convergence along shallow gradients. The momentum term allows the path the network takes toward the solution to pickup speed in the downhill direction. The error surface may consist of long gradually sloping ravines that end at a minima point. Convergence along these ravines is slow and usually the algorithm oscillates across the ravine valley as it moves towards a solution. Therefore, it is difficult to speed up the process without increasing the chance of overshooting the minima, however, the addition of the momentum term is fairly successful. This difficulty could be removed if the momentum factor were selected to be small near the minima and large far from the minima. The proper choice of μ and λ has a significant effect on the performance of BP learning algorithm. Improper choices of these parameters may result in slow convergence, paralysis and continuous instability.

Several researchers have investigated the effect of adaptation of momentum factor and steepness parameter on the performance of BP algorithm. In [6], the steepness of every neuron is adjusted such that the average distance of the two closest data points from the dividing hyper-plane is attained, while in [3], the steepness of every neuron is adjusted by gradient descent algorithm. In [7], the momentum factor is adjusted in order to cancel the introduced noise (which is the result of misadjustment of momentum factor) and to retain the speed-up as well as convergence. In [8], the momentum factor is considered as a function of gradient and in [9], the error function is divided into five regions, in which the momentum factor and learning rate are adjusted differently. In [10], the Mean Square Error (MSE) is considered as a function of the learning rate and momentum factor and these parameters are adjusted to minimize the MSE.

Often the mean-square error surfaces for back-propagation algorithm are multi-modal. The learning automaton is known to have a well-established mathematical foundation and global optimization capability [11]. This latter capability of the learning automaton can be used fruitfully to search a multi-modal mean-square error surface. Variable Structure Learning Automata (VSLA) have been used to find the appropriate value for different parameters of BP learning algorithm including learning rate [12,13], steepness parameter [2] and momentum factor [1,14]. In this paper, the application of the Fixed Structure Learning Automata (FSLA) is presented for appropriate selection of the momentum factor and steepness parameter of BP algorithm in order to achieve a higher rate of convergence and also to increase the probability of escaping the local minima. The feasibility of the proposed method is shown through simulations on three learning problems: exclusive-or, encoding problem and digit recognition. These problem are chosen because they posses different surfaces and collectively present an environment that is suitable to determine the effect of the proposed method. Simulation on these problems shows that the adaptation of momentum factor and steepness parameter using this method increases not only the convergence rate but also the likelihood of bypassing the local minima. Also simulations show that the FSLA approach for adaptation of BP parameters performs much better than the VSLA approach reported in [1,2].

The paper is organized as follows: The learning automaton is introduced in the next section and then, the proposed method is presented. Simulation results and discussion are also provided followed by the conclusion.

LEARNING AUTOMATA

Learning automata can be classified into two main families, fixed and variable structure learning automata [11]. Examples of the FSLA type, which is used in this paper, are Tsetline, Krinsky and Krylov automata. A fixed structure automaton is quintuple $< \alpha, \phi, \beta, F, G >$ where:

1. $\alpha = (\alpha_1, ..., \alpha_R)$ is the set of actions that must be chosen from;
2. $\phi = (\phi_1, ..., \phi_s)$ is the set of states;
3. $\beta = \{0, 1\}$ is the set of inputs where $\beta = 1$ represents a penalty and $\beta = 0$ a reward;
4. $F : \phi \times \beta \rightarrow \phi$ is the transition map, which defines the transition of the state of the automaton on receiving an input, F may be stochastic;

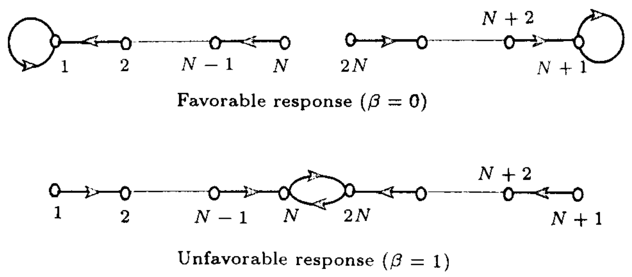


Figure 3. The state transition graph for $L_{2N,2}$.

5. $G : \phi \rightarrow \alpha$ is the output map and determines the action taken by the automaton if it is in state ϕ_j.

The selected action serves as the input to the environment which in turn emits a stochastic response $\beta(n)$ at time n. $\beta(n)$ is an element of $\beta = \{0, 1\}$ and is the feedback response of the environment to the automaton. The environment penalizes (i.e., $\beta(n) = 1$) the automaton with the penalty probability c_i, which is action dependent. On the basis of $\beta(n)$ response, the state of the automaton $\phi(n)$ is updated and a new action is chosen at (n+1). Note that c_i are unknown initially and it is desired that as a result of interaction between the automaton and the environment, the action with minimum penalty response in an expected sense is obtained. In the next few paragraphs, two action-fixed structure learning automata and a variable structure learning automaton, used in this paper, are described.

Tsetline Automaton ($L_{2N,2}$)

This automaton has $2N$ states and two actions and attempts to incorporate the past behavior of the system in its decision rule for choosing the sequence of actions. The $L_{2N,2}$ automaton keeps an account of the number of successes and failures received for each action. It is only when the number of failures exceeds the number of successes, or some maximum value N, that the automaton switches from one action to another. The procedure described above is a convenient method for keeping track of the performance of the actions α_1 and α_2. N is called the memory depth associated with each action and automaton is said to have a total memory of $2N$. The state transition graph of this automaton is shown in Figure 3. For every favorable response, the state of automaton moves deeper into the memory of the corresponding action and for an unfavorable response, moves out of it.

TsetlineG Automaton ($G_{2N,2}$)

This automaton behaves exactly like $L_{2N,2}$ automaton when the response of the environment is favorable, but for unfavorable responses it switches from state ϕ_N to ϕ_{N+1} and from state ϕ_{2N} to ϕ_1. Consequently,

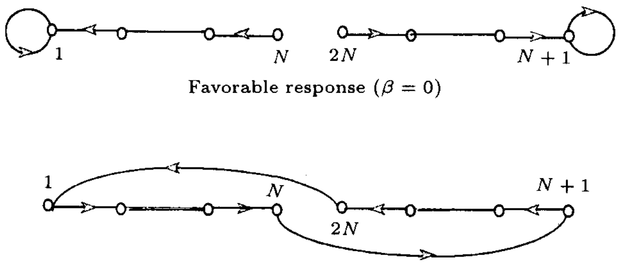

1
N
2N
N + 1
Favorable response ($\beta = 0$)
1
N
N + 1
2N
Unfavorable response ($\beta = 1$)

Figure 4. The state transition graph for $G_{2N,2}$.

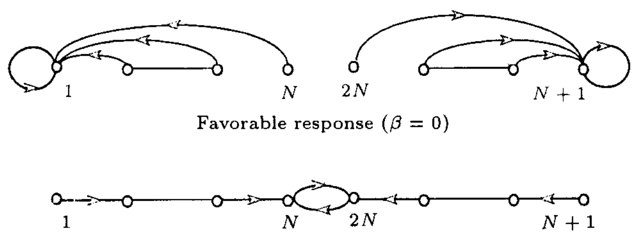

1
N
2N
N + 1
Favorable response ($\beta = 0$)
1
N
2N
N + 1
Unfavorable response ($\beta = 1$)

Figure 5. The state transition graph for Krinsky automaton.

this automaton performs an action at least N times (resulting in N consecutive unfavorable responses) before switching to another action. The state transition graph of this automaton is shown in Figure 4.

Krinsky Automaton

This automaton behaves exactly like $L_{2N,2}$ automaton when the response of the environment is unfavorable, but for favorable responses, any state ϕ_i (for $i = 1, \cdots, N$) passes to the state ϕ_1 and any state ϕ_i (for $i = N+1, \cdots, 2N$) passes to the state ϕ_{N+1}. This implies that a string of N consecutive unfavorable responses is needed to change an action to another. The state transition graph of this automaton is shown in Figure 5.

Krylov Automaton

This automaton has state transitions that are identical to the $L_{2N,2}$ automaton when the output of the environment is favorable. However, when the response of the environment is unfavorable, state ϕ_i ($i \neq 1, N, N+1, 2N$) passes to state ϕ_{i+1} or ϕ_{i-1} with the probability of 0.5. When $i = 1$ or $i = N+1$, ϕ_i stays in the same state with a probability of 0.5 and moves to ϕ_{i+1} with the same probability. When $i = N$, automaton state moves to ϕ_{N-1} or to ϕ_{2N} with the same probability of 0.5. When $i = 2N$, automaton state moves to ϕ_{2N-1} or to ϕ_N with the same probability of 0.5. The state transition graph of this automaton is shown in Figure 6.

All of the above mentioned automata can be extended to multiple action automaton.

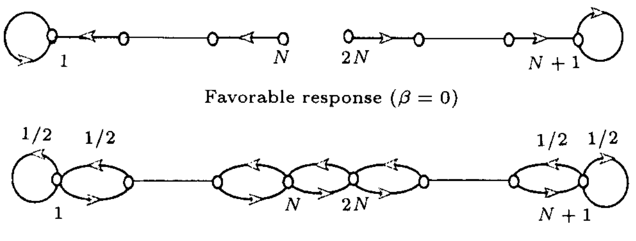

1
N
2N
N + 1
Favorable response ($\beta = 0$)
1/2
1/2
1/2
1/2
1
N
2N
N + 1
Unfavorable response ($\beta = 1$)

Figure 6. The state transition graph for Krylov automaton.

Variable Structure Learning Automata

Variable structure learning automaton is represented by sextuple $< \beta, \phi, \alpha, P, G, T >$, where β is a set of input actions, ϕ is a set of internal states, α is a set of outputs, P denotes the state probability vector governing the choice of the state at each stage k, G is the output mapping and T is learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector.

It is evident that the crucial factor affecting the performance of the variable structure learning automata is the learning algorithm for updating the action probabilities. Various learning algorithms have been reported in the literature [15]. Let α_i be the action chosen at time k as a sample realization from distribution $p(k)$. The linear reward-inaction algorithm (L_{R-I}) is one of the earliest schemes. In an L_{R-I} scheme, the recurrence equation for updating p is defined as:

$$p_j(k) = \begin{cases} p_j(k) + \theta(1 - p_j(k)) & \text{if } i = j \\ p_j(k) - \theta p_j(k) & \text{if } i \neq j \end{cases}, \tag{5}$$

if β is zero, P is unchanged if β is one. The parameter θ is called step length and determines the amount of increase (decrease) of the action probabilities.

In linear reward-penalty algorithm (L_{R-P}) scheme, the recurrence equation for updating p is defined as:

$$p_j(k+1) = \begin{cases} p_j(k) + \theta(1 - p_j(k)) & \text{if } i = j \\ p_j(k) - \theta p_j(k) & \text{if } i \neq j \end{cases} \quad \beta = 0, \tag{6}$$

$$p_j(k+1) = \begin{cases} p_j(k)(1-\gamma)) & \text{if } i = j \\ \frac{\gamma}{R-1} + (1-\gamma)p_j(k) & \text{if } i \neq j \end{cases} \quad \beta = 1. \tag{7}$$

The parameters θ and γ represent step lengths and determine the amount of increase (decreases) in the action probabilities.

Learning automata have been used in many applications such as: graph partitioning [16], graph isomorphism [17], optimization of neural network structures [18,19], cellular learning automata [20], queuing theory [21], telephone traffic control [22] and pattern recognition [23] (for more information on learning automata, refer to [11,15,24-28]).

THE PROPOSED METHOD

In the method proposed here, the fixed-structure learning automaton is used for adjusting the momentum factor and steepness parameter. The interconnection of learning automaton and neural network is shown in Figure 7. The neural network is the environment for the learning automaton. The learning automaton, according to the amount of error received from the neural network, adjusts the values of parameters of the backpropagation algorithm. The actions of the automaton correspond to the values of the momentum factor (or steepness parameter) and input to the automata is a function of error in the output of neural network.

At the beginning of each epoch of BP algorithm, the learning automaton selects one of its actions (in the fixed-structure learning automaton, action is selected by means of output function G and in variable structure learning automaton, the action is selected by a sample realization of probability vector p). The value of selected action is used in BP algorithm for that epoch. The response of the environment, which is given to the learning automaton, is a function of the mean square error as explained below.

In the kth epoch, the average of mean square error in past W epochs is computed by the following equation (W is called the window size):

$$MSE_W(k) = \frac{1}{W} \sum_{m=1}^{W} MSE(k-m), \tag{8}$$

where $MSE(n)$ and $MSE_W(k)$ denote mean square error in the nth epoch and average mean square error in the past W epochs, respectively. Then, $MSE(k)$ is compared with $MSE_W(k)$ and if $MSE_W(k)-MSE(k)$ is less than a threshold, the automaton receives a penalty from the environment, otherwise it receives a reward. Now, the response of the environment (input to the learning automaton) can be formulated as follows:

$$\beta(n) = \begin{cases} 0 & \text{if } MSE_W(n) - MSE(n) \le T \\ 1 & \text{if } MSE_W(n) - MSE(n) > T \end{cases} \tag{9}$$

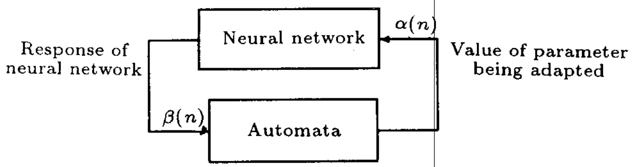


Figure 7. The interconnection of learning automaton and neural network.

At the beginning of the first epoch, the action of the learning automaton is selected randomly from the set of allowable actions.

The algorithms presented later in this paper are backpropagation algorithms in which the learning automaton is responsible for the adaptation of the BP parameters. In such an algorithm, at each iteration, one input of the training set is presented to the neural network, then the network response is computed and the weights are corrected. The weight correction is applied at the end of each epoch. The amount of correction is proportional to the BP parameters. Using the learning automaton as an adaptation technique, the search for optimum values for the BP parameters is carried out in probability space rather than parameter space and, therefore, the algorithm is provided with the ability to locate the global optimum.

Algorithms of Figures 8 and 9 describe how fixed-

```
Procedure One LA-BP Algorithm
    Initialize the weights to small random values
    Initialize the parameters of LA
    repeat
        for all training pairs $(X, T)$ do
            call FeedForward
            call ComputeGradiant
        end for
        call UpdateWeights
        call Response (Network)
        call Adjust-BP-Parameter
    until termination condition is satisfied
end procedure
```

Figure 8. Backpropagation algorithm with a single learning automaton.

```
Procedure Two LA-BP Algorithm
    Initialize the weights to small random values
    Initialize the parameters of LA
    repeat
        for all training pairs $(X, T)$ do
            call FeedForward
            call ComputeGradiant
        end for
        call UpdateWeights
        call Response (Hidden Layer)
        call Adjust-BP-Parameter (Hidden Layer)
        call Response (Output Layer)
        call Adjust-BP-Parameter (Output Layer)
    until termination condition is satisfied
end procedure
```

Figure 9. Backpropagation algorithm with two learning automaton.

structure learning automata can be used for determination of momentum factor and steepness parameter of backpropagation algorithm. In the first algorithm, a single learning automaton is responsible for determination of the BP parameter for the whole network, whereas in the second algorithm, a separate learning automaton has been used for each layer (hidden and output layers). Simulation results demonstrate that by using separate learning automaton for each layer of the network not only does the performance of the network improve over the case where only a single automaton is used, but also the likelihood of bypassing the local minima increases. These two algorithms have been tested with respect to several problems and the results are presented in the next section.

Simultaneous Adaptation of Momentum Factor and Steepness Parameter

The rate of convergence and the stability of the training algorithm can be improved if both momentum factor and steepness parameter are adapted simultaneously. The following two algorithms (Figures 10 and 11) describe the simultaneous adaptation of momentum factor and steepness parameter.

In the first algorithm, the network uses one automaton to adjust the momentum factor and another automaton to adjust the steepness parameter. Both automata work simultaneously to adjust the momentum factor and steepness parameter.

In the second algorithm, the network uses two pairs of automata, the first pair of automata (one for each layer) is responsible for adjusting the steepness parameter and the second pair (one for each layer) is responsible for adjusting the momentum factor. These four automata work simultaneously to adapt steepness parameter and momentum factor. The algorithms have been tested with several problems and the results are presented in the next section.

```
Procedure Simultaneous-LA-BP Algorithm
    Initialize the weights to small random values
    Initialize the parameters of LA
    repeat
        for all training pairs $(X, T)$ do
            call FeedForward
            call ComputeGradiant
        end for
        call UpdateWeights
        call Response (Network)
        call Adjust (Steepness Parameter)
        call Adjust (Momentum Factor)
    until termination condition is satisfied
end procedure
```

Figure 10. Backpropagation algorithm for simultaneous adjustment of momentum factor and steepness parameter.

```
Procedure Simultaneous-Two-LA-BP Algorithm
    Initialize the weights to small random values
    Initialize the parameters of LA
    repeat
        for all training pairs $(X, T)$ do
            call FeedForward
            call ComputeGradiant
        end for
        call UpdateWeights
        call Response (Hidden Layer)
        call Adjust (Hidden layer steepness parameter)
        call Adjust (Hidden layer momentum factor)
        call Response (Output Layer)
        call Adjust (Output layer steepness parameter)
        call Adjust (Output layer momentum factor)
    until termination condition is satisfied
end procedure
```

Figure 11. Backpropagation algorithm for simultaneous adjustment of momentum factor and steepness parameter when each layer has its own automaton.

SIMULATION

In order to evaluate the performance of the proposed method, simulations are carried out on three learning problems: exclusive-or, encoding problem and digit recognition. The results are compared with those obtained from the standard BP and variable structure learning automata based algorithm reported in [1,2,12,14]. These problems are chosen due to possessing different error surfaces and collectively presenting an environment that is suitable for determining the effect of the proposed method. Actions of the learning automata in these simulations are selected in [0,1] interval with equal distance, i.e., the value of the ith action of the learning automaton with K actions is chosen to be $\frac{i}{K}$. For the sake of convenience in presentation, automaton (K, N) is used to refer to a fixed structure learning automaton, with K actions and memory depth of N. For all simulations reported in this paper, the same values for BP parameters are used in all experimentation of different algorithms, except for parameters which are being adapted by the algorithm.

XOR

The network architecture used for solving this problem consists of 2 input units, 2 hidden units and 1 output unit [5]. Figure 12 illustrates the effectiveness of using FSLA and VSLA on the adaptation of momentum factor and Figure 10 shows the effectiveness of FSLA and VSLA regarding the adaptation of steepness parameter. For automaton in Figure 13, the threshold of 0.001 and window size of 1 and in Figure 10, the threshold of 0.0001 and window size of 1 are chosen.

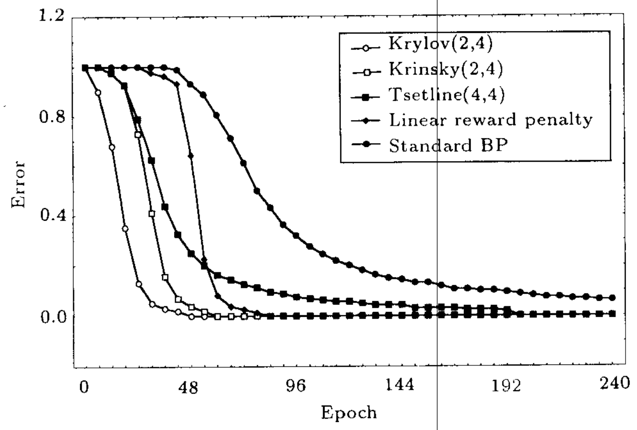


Figure 12. Adaptation of momentum factor for XOR problem when a single learning automaton is used.

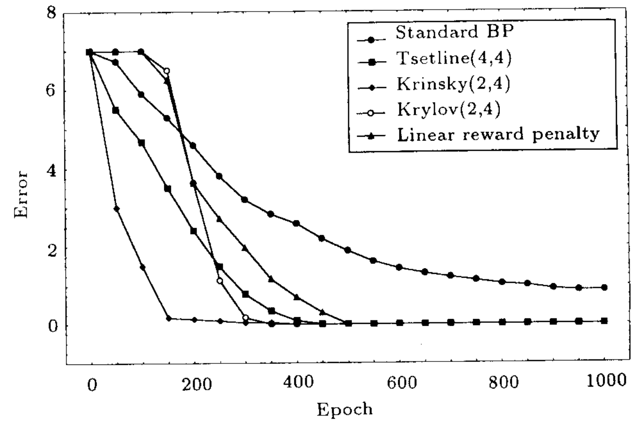


Figure 14. Adaptation of momentum factor for encoding problem when a single learning automaton is used.

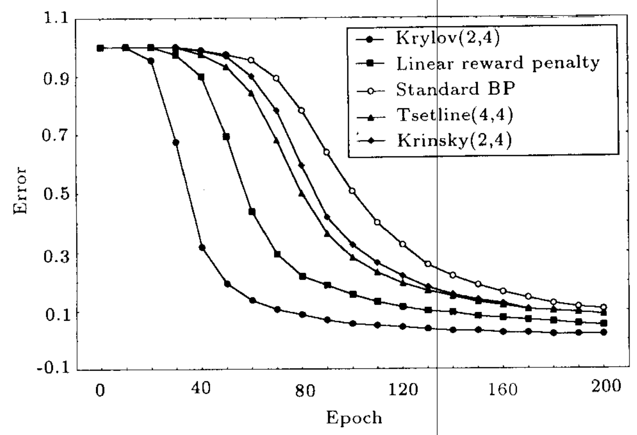


Figure 13. Adaptation of steepness parameter for XOR problem when a single learning automaton is used.

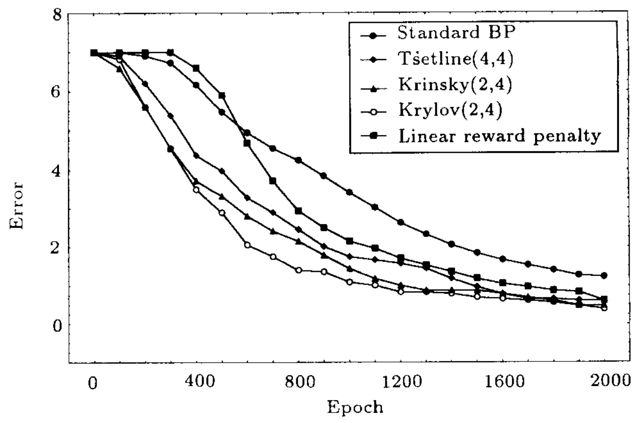


Figure 15. Adaptation of steepness parameter for encoding problem when a single learning automaton is used.

For linear reward-penalty automaton, the reward and penalty coefficients are 0.001 and 0.0001, respectively.

Encoding Problem

In this problem, a set of orthogonal input patterns are mapped to a set of orthogonal output patterns through a small set of hidden units [5]. The network architecture used for solving this problem consists of 8 input units, 3 hidden units and 8 output units. Figures 14 and 15 show the effectiveness of using FSLA and VSLA regarding the adaptation of momentum factor and steepness parameter, respectively. For automata in these figures, the threshold of 0.01 and window size of 1 are considered. For linear reward-penalty automaton, the reward and penalty coefficients are 0.001 and 0.0001, respectively.

8 × 8 Dot Numeric Font Learning

Ten numbers, 0, ... ,9, each represented by a 8 × 8 grid of black and white dots are considered, as shown in Figure 16 [3]. The network must learn to distinguish

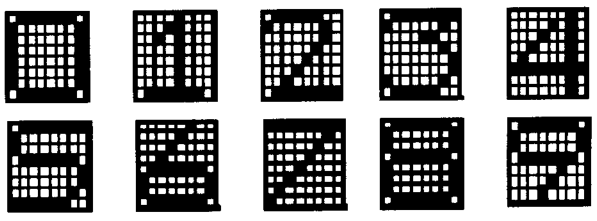


Figure 16. The training set of neural network for digit problem.

these classes. For this problem, the network consists of 64 input units connected to 6 hidden units and 10 output units.

Figures 17 and 18 illustrate the effectiveness of using FSLA and VSLA regarding the adaptation of momentum factor and steepness parameter, respectively. For automata in these figures, the threshold of 0.01 and window size of 1 are chosen. For linear reward-penalty automaton, the reward and penalty coefficients are 0.001 and 0.0001, respectively.

The results obtained when a single learning au-

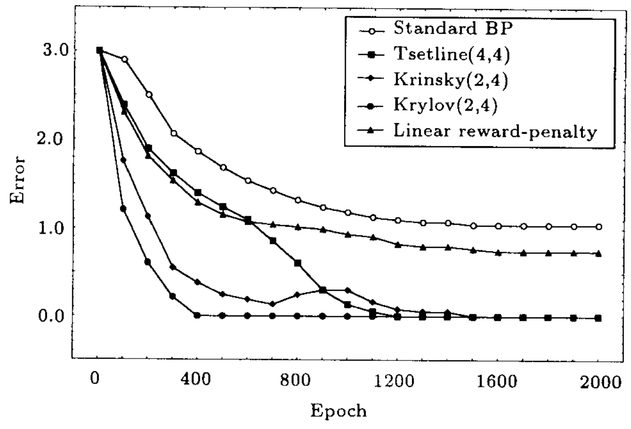


Figure 17. Adaptation of momentum factor for digit problem when a single learning automaton is used.

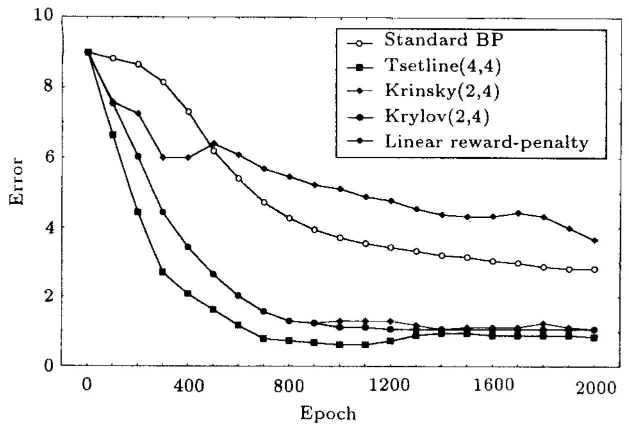


Figure 18. Adaptation of steepness parameter for digit problem when a single learning automaton is used.

tomaton is assigned to the whole network for adjusting the momentum factor are presented in Table 1. For all automata in this simulation, the threshold of 0.01 and window size of 1 are chosen. The error of standard BP after 5000 epochs is 0.1675668 .

Table 2 shows the results obtained when a single learning automaton is assigned to the whole network for adjusting the steepness parameter. For all automata in this simulation, the threshold of 0.01 and window size of 6 are chosen. The error of BP with constant steepness parameter after 5000 epochs is 0.1017534.

Table 3 depicts the performance of the network when different automata are assigned to different layers

Table 1. Simulation results for digit problem when momentum factor is adapted (single learning automaton).

Learning Automata	**Final Mean Square Error**	**Epochs for Error Goal = 0.01**
Tsetline (4, 4)	0.0099858	3289
TsetlineG (2,4)	0.0099850	564
Krinsky (2, 4)	0.0099918	1101
Krylov (2, 4)	0.0099962	879

Table 2. Simulation results for digit problem when steepness parameter is adapted (single learning automaton).

Learning Automata	**Final Mean Square Error**	**Epochs for Error Goal = 0.01**
Tsetline (4, 4)	0.0099855	4560
TsetlineG (2, 4)	0.0074391	2886
Krinsky (2, 4)	0.0074391	2886
Krylov (2, 4)	0.0099975	4005

Table 3. Simulation results for digit problem when momentum factor is adapted (one learning automaton for each layer).

Hidden Layer Automata	**Output Layer Automata**	**Final Mean Square Error**	**Epochs for Error = 0.01**
Tsetline (4, 4)	Tsetline (4, 4)	0.0099984	3010
Tsetline (4, 4)	Krinsky (2, 4)	0.0089071	2407
Tsetline (4, 4)	Krylov(2, 4)	0.0099510	775
Tsetline (4, 4)	TsetlineG(2, 4)	0.0099897	622
Krinsky(2, 4)	Tsetline (4, 4)	0.0099577	112
Krinsky(2, 4)	Krinsky(2, 4)	0.0099736	649
Krinsky(2, 4)	Krylov(2, 4)	0.0099965	789
Krinsky(2, 4)	TsetlineG(2, 4)	0.0099998	822
Krylov(2, 4)	Tsetline (4, 4)	0.0099859	899
Krylov(2, 4)	Krinsky(2, 4)	0.0099850	417
Krylov(2, 4)	Krylov(2, 4)	0.0099891	880
Krylov(2, 4)	TsetlineG(2, 4)	0.0099806	614
TsetlineG(2, 4)	Tsetline (4, 4)	0.0099882	113
TsetlineG(2, 4)	Krinsky(2, 4)	0.0099924	801
TsetlineG(2, 4)	Krylov(2, 4)	0.0099941	635
TsetlineG(2, 4)	TsetlineG(2, 4)	0.0099970	895

to adapt the momentum factor. Each automaton is responsible for adaptation of momentum factor for its assigned layer. For all automata in this simulation, the threshold of 0.01 and window size of 1 are chosen. The error of BP with constant momentum factor after 5000 epochs is 0.1675668.

Table 4 presents the results of the experiments in which one automaton is used by each layer of the network to adjust the steepness parameter of that layer. For all automata in this simulation, the threshold of 0.01 and window size of 6 are chosen. The error of standard BP after 5000 epochs is 0.1017534.

Remark 1

Assuming that the region within which the adapted parameter changes is fixed, by increasing the number of actions, the difference between the values of two consecutive actions are reduced and as a result, the parameter being adapted changes smoothly. This leads to less oscillation on error function, which itself

Table 4. Simulation results for digit problem when steepness parameter is adapted (one learning automaton for each layer).

Hidden Layer Automata	Output Layer Automata	Error After 5000 Epochs	Epochs For Error of 0.01
Tsetline (4, 4)	Tsetline (4, 4)	0.0099448	2969
Tsetline (4, 4)	Krinsky (2, 4)	0.0089643	2477
Tsetline (4, 4)	Krylov(2, 4)	0.0014180	3656
Tsetline (4, 4)	TsetlineG(2, 4)	0.0098581	4949
Krinsky(2, 4)	Tsetline (4, 4)	0.0990697	5000
Krinsky(2, 4)	Krinsky(2, 4)	0.1082491	5000
Krinsky(2, 4)	Krylov(2, 4)	0.0006661	3439
Krinsky(2, 4)	TsetlineG(2, 4)	0.1110588	5000
Krylov(2, 4)	Tsetline (4, 4)	0.0087339	2317
Krylov(2, 4)	Krinsky(2, 4)	0.0097019	1726
Krylov(2, 4)	Krylov(2, 4)	0.0796143	5000
Krylov(2, 4)	TsetlineG(2, 4)	0.0029425	3983
TsetlineG(2, 4)	Tsetline (4, 4)	0.0098257	1384
TsetlineG(2, 4)	Krinsky(2, 4)	0.0097729	1355
TsetlineG(2, 4)	Krylov(2, 4)	0.0095585	4715
TsetlineG(2, 4)	TsetlineG(2, 4)	0.0098836	2520

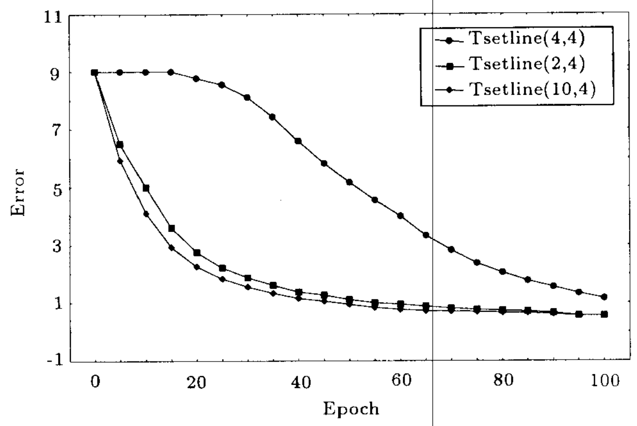


Figure 19. Effect of number of actions on the speed of convergence for digit problem when a single learning automaton is used.

results in a higher rate of convergence of the training algorithm. Figure 19 shows the effect of a number of actions on speed of convergence for the digit recognition problem. For this simulation, the window size of 10, threshold value of 0.01 and Tsetline automaton with memory depth of 4 are chosen.

Remark 2

As the memory depth increases, the selected action must be more penalized in order for the automaton to change this action. Therefore, by increasing the memory depth, the probability that the automaton chooses a wrong action will be reduced. This may lead to speeding up the convergence of the algorithm. Figure 20 shows the effect of memory depth on the speed of convergence for a digit recognition problem. For this simulation, the window size of 10, threshold value of 0.01 and Tsetline automaton with 4 actions are chosen.

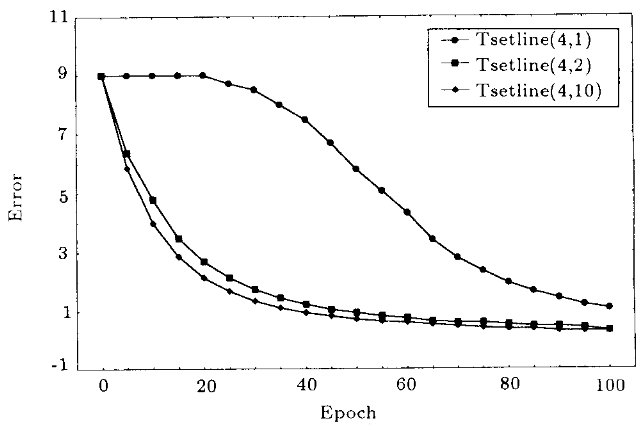


Figure 20. Effect of memory depth on the speed of convergence for digit problem when a single learning automaton is used.

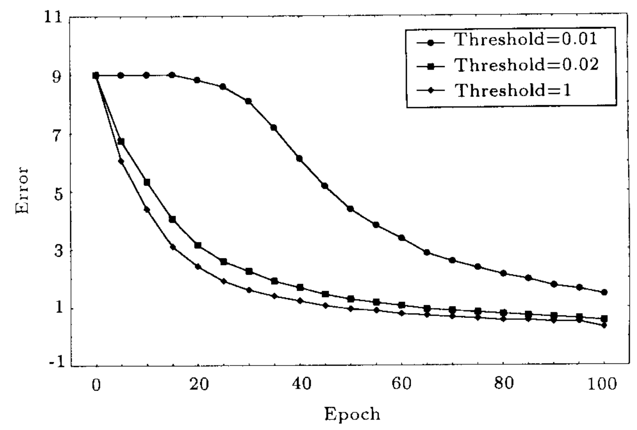


Figure 21. Effect of threshold value on the speed of convergence for digit problem when a single learning automaton is used.

Remark 3

Increasing the value of the threshold, the probability of penalizing a given action will increase. This causes the learning automaton to change its action more quickly, which enables it to find a better parameter for the region of the error surface that is being searched. Figure 21 illustrates the effect of threshold value on the speed of convergence for the digit recognition problem. For this simulation, the window size of 10 and Tsetline (4,4) automaton are used.

Remark 4

If the window size is small, then the response of the environment (neural network) is based on the information of a small region of the error surface, which may not

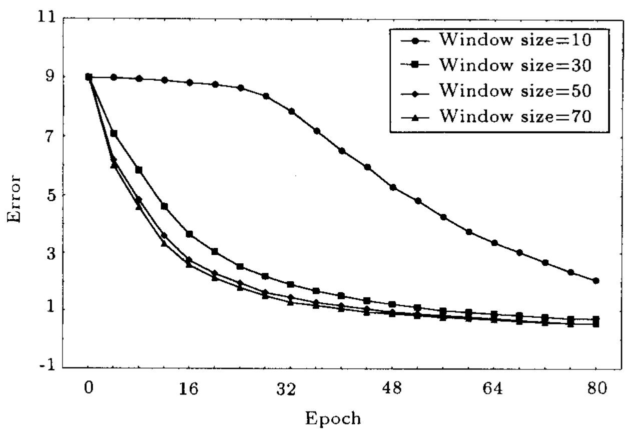


Figure 22. Effect of window size on the speed of convergence for digit problem when a single learning automaton is used.

well characterize the whole error surface. By increasing the window size, the algorithm uses information from a larger region of the error surface for parameter adaptation. This may lead to a better performance. Figure 22 depicts the effect of window size on the speed of convergence for the digit recognition problem. For this simulation, the threshold of 0.01 and Tsetline (4,4) automaton are used.

Remark 5

In the first order methods of gradient descent (such as BP), weights are adapted in the negative direction of the gradient vector. In this approach, the path from the initial point (initial weight) to minimum may follow a zigzag path. The adaptation in ith iteration may spoil the adaptation in $(i-1)$th iteration. The zigzag path slows down the speed of convergence of the algorithm. The conjugate-gradient method avoids the low speed of convergence by incorporating the direction vector (direction in which the weights are adapted) and gradient vector. Assuming that $q(n)$ denotes the direction vector at nth iteration, the weight vector w of neural network is updated by the following equation:

$$\Delta w_{jk}(n+1) = \alpha(n) \times q_{jk}(n), \tag{10}$$

where $\alpha(n)$ is the learning rate in iteration n and chosen in such a way to minimize the error surface along the direction vector $q(n)$, computed by the following equation:

$$\alpha(n) = arg\min_{\alpha} MSE(w(n) + \alpha q(n)). \tag{11}$$

The initial value of direction vector is the negative of gradient vector. In $(n+1)$th iteration, the direction vector is computed as a linear combination of gradient vector and previous direction vector given by:

$$q_{jk}(n+1) = -\frac{\partial E}{\partial w_{jk}} + \Psi(n)q(n), \tag{12}$$

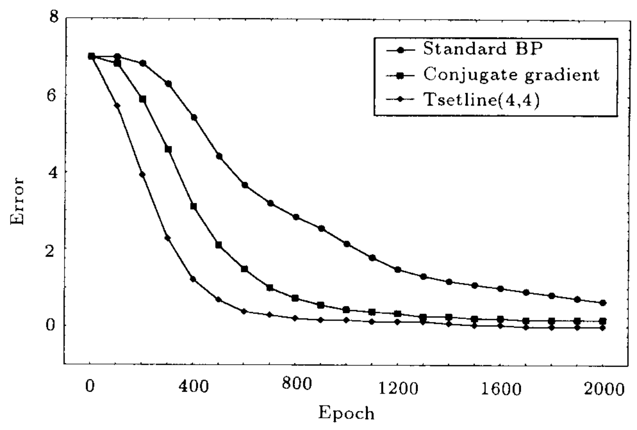


Figure 23. Comparison of conjugate gradient method and adaptation of momentum factor using learning automata for encoding problem.

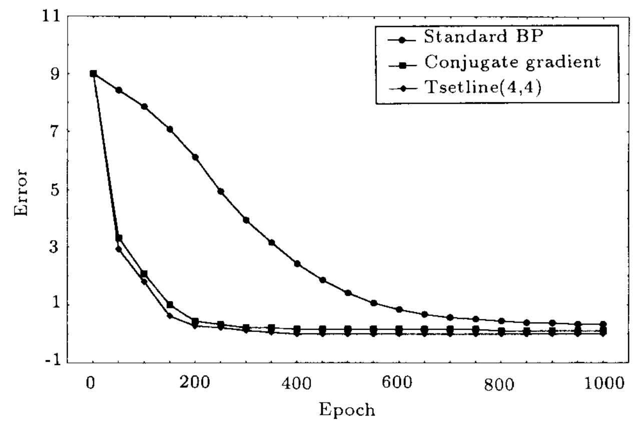


Figure 24. Comparison of conjugate gradient method and adaptation of momentum factor using learning automata for digit problem.

where $\Psi(n)$ is time varying parameter computed as:

$$\Psi(n) = \frac{[\frac{\partial E}{\partial w}(n+1)]^T[\frac{\partial E}{\partial w}(n+1)]}{[\frac{\partial E}{\partial w}(n)]^T[\frac{\partial E}{\partial w}(n)]}. \tag{13}$$

Figures 23 to 26 compare the performance of conjugate-gradient method with the proposed scheme when the momentum factor or steepness parameter is adapted for encoding and digit recognition problems. For these simulations, the threshold of 0.01 and window size of 1 are used.

Remark 6

Baba and Handa [14] have used a similar approach for adaptation of the momentum factor of BP algorithm. In their method, a hierarchical structure learning automaton has been employed instead of a single learning automaton in order to automatically choose an appropriate momentum factor. In this approach, the learning automaton selects a momentum

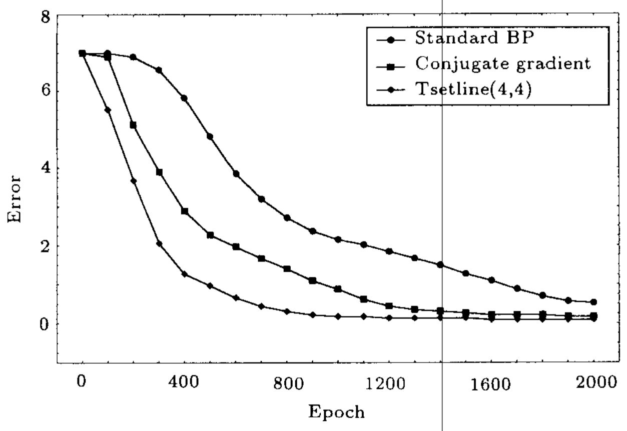


Figure 25. Comparison of conjugate gradient method and adaptation of steepness parameter using learning automata for encoding problem.

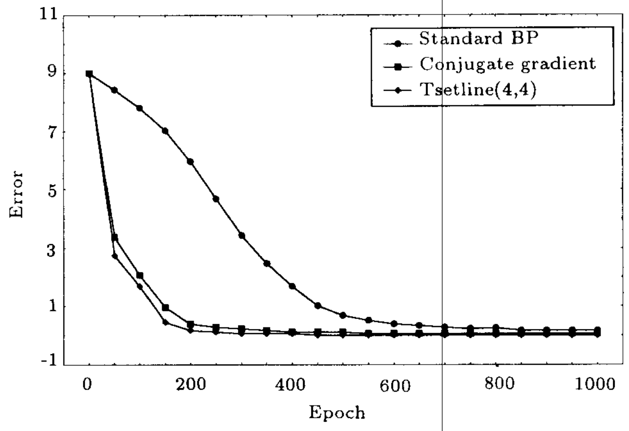


Figure 26. Comparison of conjugate gradient method and adaptation of steepness parameter using learning automata for digit problem.

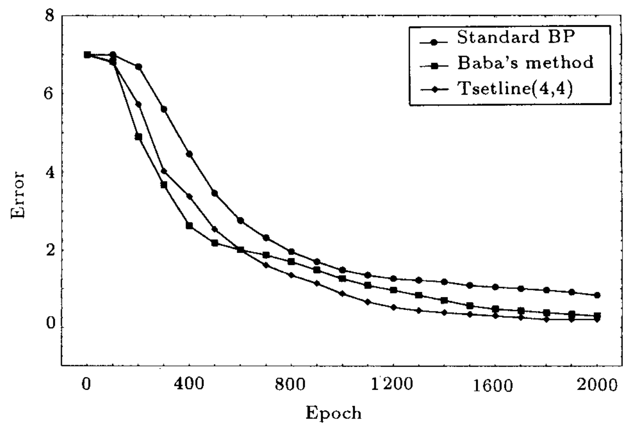


Figure 27. Comparison of different learning automata based methods for adaptation of momentum factor for encoding problem.

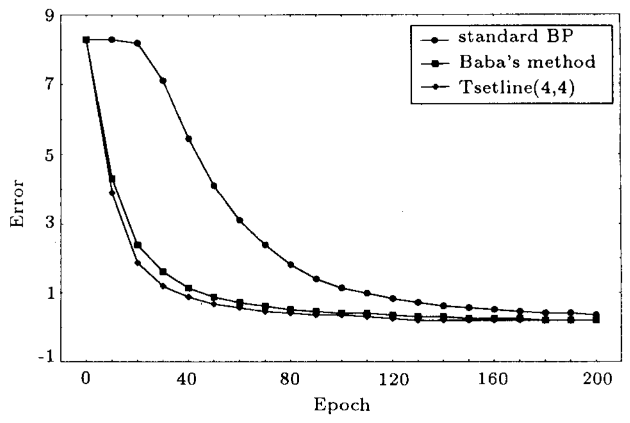


Figure 28. Comparison of different learning automata based methods for adaptation of momentum factor for digit problem.

factor and BP algorithm uses this parameter to adjust the weights of neural network in an epoch. At the end of every epoch, the learning automaton receives a reward if the mean square error is decreased and receives a penalty otherwise. In Figures 27 and 28, the performance of Baba method has been compared with the proposed scheme for encoding and digit recognition problems, when the momentum factor is adapted. For this simulation, the threshold of 0.01 and window size of 1 are used. The parameters of Baba method are the same as parameters used in their previous work [14]. As shown in Figures 27 and 28, the proposed scheme exhibits a higher performance compared with the Baba method.

Careful inspection of Baba algorithm reveals that it is a special case of the algorithm proposed here when window size is 1 and threshold value is 0. The proposed algorithm is superior to Baba scheme because: 1) Advantages of window size and threshold value have not been considered in Baba algorithm, 2) In Baba scheme, hierarchical automaton has been used which makes the algorithm more complicated and time consuming, 3) Baba method is proposed for adaptation of momentum factor only.

Remark 7

Adaptive steepness (ASBP) method [3] is a method which uses gradient descent rule for adaptation of steepness parameter. In this method, each neuron k has a steepness parameter λ_k, which is changed by the following rule:

$$\Delta\lambda_k = -\epsilon\frac{\partial E}{\partial \lambda_k}. \tag{14}$$

Figures 29 and 30 compare the performance of ASBP and proposed schemes for encoding and digit recognition problems, respectively. For these simulations, the threshold of 0.01 and window size of 1 are chosen.

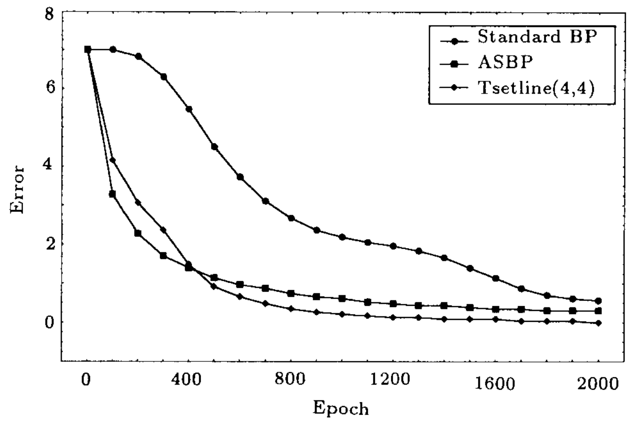


Figure 29. Comparison of different methods for adaptation of steepness parameter for encoding problem.

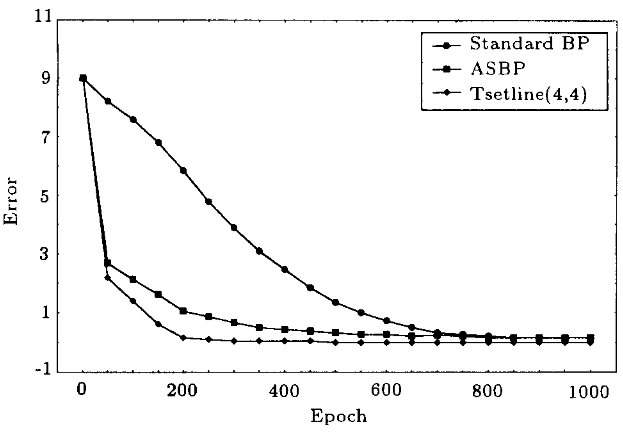


Figure 30. Comparison of different methods for adaptation of steepness parameter for digit problem.

Simulation Results for Simultaneous Adaptation

In another experiment, whose results are presented in Table 5, two automata of the same kind are used to adapt momentum factor and steepness parameter simultaneously. As seen in this table, the best result is obtained for the case when two Krinsky automata are used for adaptation of both steepness parameter and momentum factor. For these experiments, the threshold of 0.01 and window size of 3 are chosen. The error of BP after 5000 epochs is 0.0976520.

Table 6 shows the effect of associating different automata to different layers of the network for adjusting the momentum factor and steepness parameter simultaneously. The same pair of automata are used for adjusting both steepness parameter and momentum factor. It can be seen that the best pair of automata which gives the highest rate of convergence is Krylov for hidden layer and Tsetline for output layer. Note that this pair of automata is used for adaptation of momentum factor as well as steepness parameter. For all automata in these experiments, the threshold of 0.01 and window size of 3 are chosen. The error of BP with constant momentum factor after 5000 epochs is 0.0976520.

Table 5. Simultaneous adaptation of momentum factor and steepness parameter for digit problem when a single learning automaton is used for the whole of the network.

Learning Automata	Final Mean Square Error	Epochs for Error
Tsetline (4, 4)	0.0092784	807
TsetlineG (2, 4)	0.099826	982
Krinsky (2, 4)	0.0084366	49
Krylov (2, 4)	0.0098485	699

Table 6. Simultaneous adaptation of momentum factor and steepness parameter for digit problem when a single learning automaton is used for each layer.

Hidden Layer Automata	Output Layer Automata	Error	Epochs
Tsetline (4, 4)	Tsetline (4, 4)	0.0098906	1822
Tsetline (4, 4)	Krinsky (2, 4)	0.0099240	1124
Tsetline (4, 4)	Krylov(2, 4)	0.0094257	1614
Tsetline (4, 4)	TsetlineG(2, 4)	0.0099214	814
Krinsky(2, 4)	Tsetline (4, 4)	0.0036989	2076
Krinsky(2, 4)	Krinsky(2, 4)	0.0099466	566
Krinsky(2, 4)	Krylov(2, 4)	0.0070825	1187
Krinsky(2, 4)	TsetlineG(2, 4)	0.0095368	62
Krylov(2, 4)	Tsetline (4, 4)	0.0050398	31
Krylov(2, 4)	Krinsky(2, 4)	0.0098945	1303
Krylov(2, 4)	Krylov(2, 4)	0.0099984	386
Krylov(2, 4)	TsetlineG(2, 4)	0.0099983	724
TsetlineG(2, 4)	Tsetline (4, 4)	0.0081699	1758
TsetlineG(2, 4)	Krinsky(2, 4)	0.0099911	606
TsetlineG(2, 4)	Krylov(2, 4)	0.0098906	131
TsetlineG(2, 4)	TsetlineG(2, 4)	0.0099330	780

Remark 8

In this remark, the ability of the proposed algorithm to escape local minima is considered. For this purpose, a problem in which local minima occurs frequently is chosen [29]. The training set of this problem is given in Table 7. The network considered has two input nodes x and y, two hidden units and one output unit. In this problem, if hidden units produce lines a and b, a local

Table 7. Training set for a given problem.

Pattern	x	y	Desired Output
A	0	0	0
B	1	0	1
C	1	1	0
D	0	1	1
E	0.5	0.5	0

minima will occur and if hidden units produce lines *c* and *d*, a global minima will occur [30]. Figure 31 shows these configurations.

The error surface of the given network as a function of weights $W_{2,1,1}$ and $W_{1,1,1}$ is given in Figure 32.

In each simulation, the same initial points for both BP (with constant momentum) and the proposed algorithm are chosen to adapt the steepness parameter. Simulations have been carried out for 20 runs. Each run uses a random initial point near the local minima. In these simulations, the BP stuck at local minima for all 20 runs whereas the proposed algorithm escaped local minima for 13 runs and stuck at local minima for 7 runs. Figures 33 to 35 show several sample runs for both BP and the proposed algorithm. In these figures, the initial point is denoted by letter 'B' and the converged point is denoted by letter 'A'. The curves

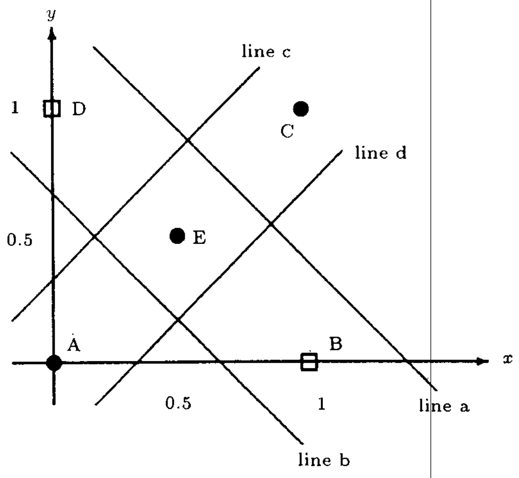


Figure 31. Lines produced by hidden units of neural network.

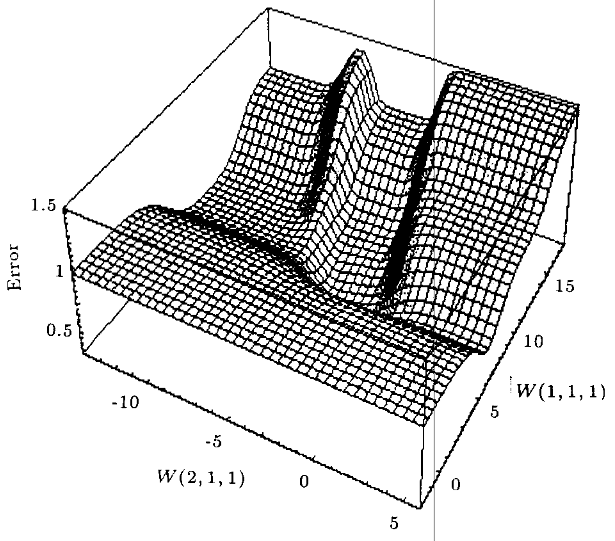
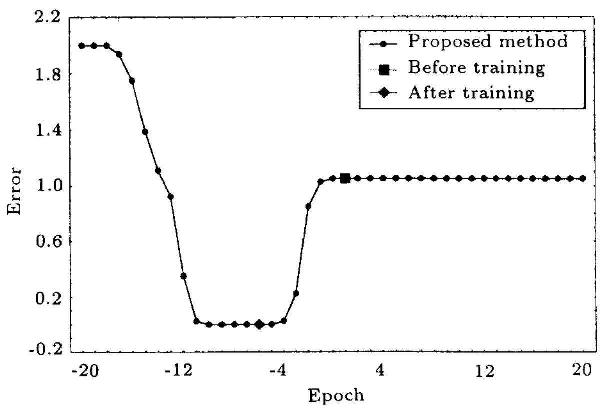


Figure 32. Error surface as a function of weights $W_{2,1,1}$ and $W_{1,1,1}$.



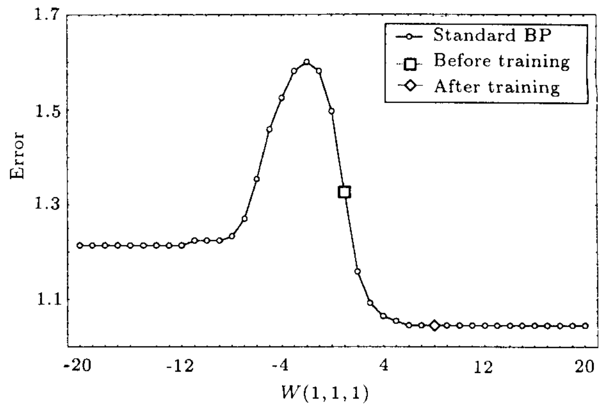
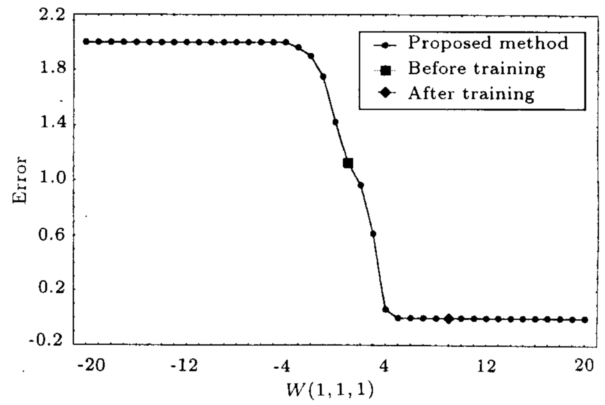


Figure 33. Sample run 1.



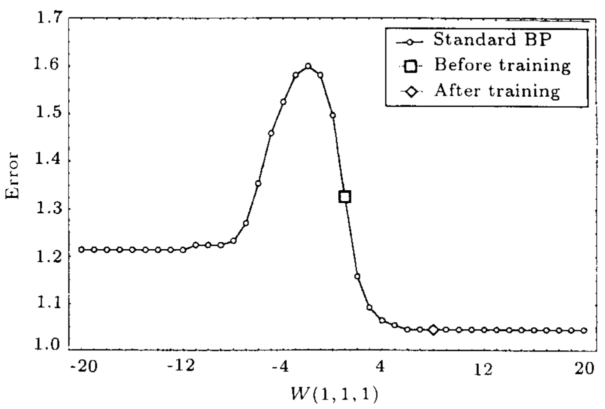


Figure 34. Sample run 2.

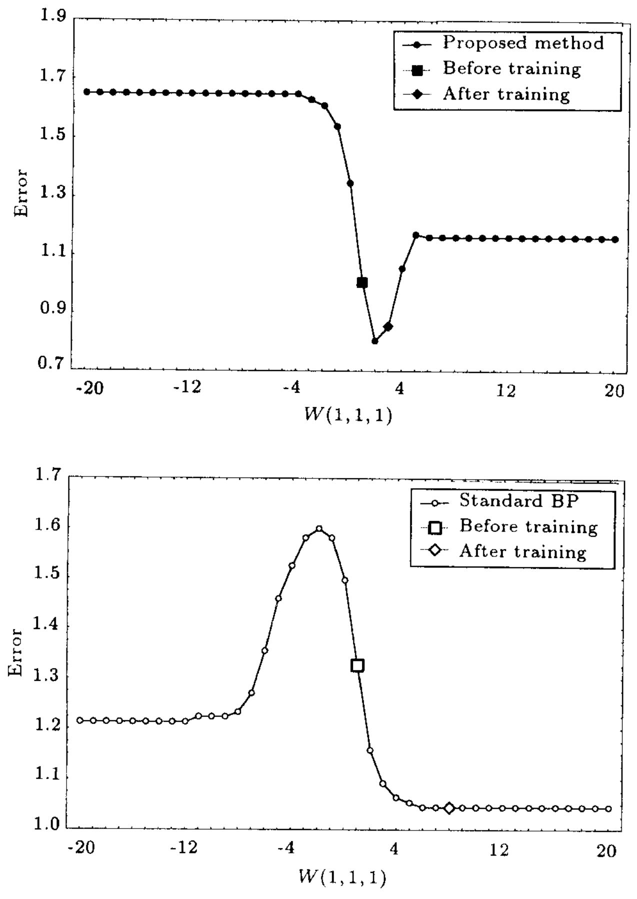


Figure 35. Sample run 3.

in these figures are obtained by projecting the error surface on axis $W_{1,1,1}$.

Table 8 shows the mean square error of the proposed method when used to adapt steepness parameter and standard BP algorithm after 10000 epochs for this problem.

CONCLUSIONS

In this paper, a fixed structure learning automaton is applied for adjusting the parameters of the BP algorithms based on the observation of the random response of a neural network. It has been demonstrated through simulations that the use of fixed-structure learning automata for adaptation of momentum factor and steepness parameter of a BP algorithm increases the rate of convergence by a large amount. By using fixed-structure learning automaton in a BP algorithm, it is possible to compute a new point that is closer to the optimum than the point computed by the algorithm itself. In all the problems studied so far, the convergence of BP, which uses fixed-structure learning automata or variable structure learning automata for adaptation of momentum factor or steepness parameter, have been higher than that of the standard BP. Simulation results also indicate that the speed of convergence can be improved if both momentum factor and steepness parameter are adapted simultaneously (Tables 5 and 6). It should be mentioned that for almost all the experiments conducted, the FSLA approach has yielded better results than that of the VSLA approach when used for adaptation of steepness parameter and momentum factor. The result of this paper can be generalized for application multi-layer neural networks.

Table 8. Mean square error after 10000 epochs.

	Standard BP		**Proposed Algorithm**	
Run	**Error**	**Converged Point**	**Error**	**Converged Point**
1	0.97	Local Minimum	0.01	Global Minimum
2	0.96	Local Minimum	0.66	Local Minimum
3	0.96	Local Minimum	0.01	Global Minimum
4	0.96	Local Minimum	0.01	Global Minimum
5	0.97	Local Minimum	0.01	Global Minimum
6	0.96	Local Minimum	0.66	Local Minimum
7	0.96	Local Minimum	0.01	Global Minimum
8	0.96	Local Minimum	0.66	Local Minimum
9	0.97	Local Minimum	0.01	Global Minimum
10	0.97	Local Minimum	0.66	Local Minimum
11	0.96	Local Minimum	0.68	Local Minimum
12	0.97	Local Minimum	0.01	Global Minimum
13	0.97	Local Minimum	0.67	Local Minimum
14	0.96	Local Minimum	0.01	Global Minimum
15	1.05	Local Minimum	0.98	Local Minimum
16	0.98	Local Minimum	0.01	Global Minimum
17	0.98	Local Minimum	0.01	Global Minimum
18	0.96	Local Minimum	0.01	Global Minimum
19	0.96	Local Minimum	0.01	Global Minimum
20	0.97	Local Minimum	0.01	Global Minimum

REFERENCES

1. Menhaj, M.B. and Meybodi, M.R. "Using learning automata in backpropagation algorithm with momentum", Technical Report, Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran (1997).

2. Menhaj, M.B. and Meybodi, M.R. "Flexible sigmodal type functions for neural nets using game of automata", *Proc. of Second Annual CSI Computer Conference CSIC-96*, Tehran, Iran, pp 221-232 (1996).

3. Sperduti, A. and Starita, A. "Speed up learning and network optimization with extended backpropagation", *Neural Networks*, **6**, pp 365-383 (1993).

4. Eaton, H.A.C. and Oliver, T.L. "Learning coefficient dependence on training set size", *Neural Networks*, **5**, pp 283-287 (1992).

5. Rumelhart, D.E., Hinton, G.E. and Williams, R.J. "Learning internal representations by error propagation", in *Parallel Distributed Processing*, Cambridge, MA: MIT Press (1986).

6. McLean, D., Bandar, Z. and ÓShea, J. "Improved interpolation and extrapolation from continuous training examples using a new neuronal model with adaptive steepness", *Proc. of ANZIIS- 94*, pp 125-129 (1994).

7. Jia, Y. and Dali, Y. "Analysis of the misadjustment of BP network and an improved algorithm", *Proc. of IEEE Int. Symp. on Circuits and Systems, ISCAS-93*, pp 2592-2595 (1993).

8. Dali, Y. and Zemin, L. "A LMS algorithm with stochastic momentum factor", *Proc. of IEEE Int. Symp. on Circuits and Systems, ISCAS-93*, pp 1250-1253 (1993).

9. Guan, L., Cheng, S. and Zhou, R. "Artificial neural network power system stabilizer trained with an improved BP algorithm", *IEE Proc. of Generation, Transmission and Distribution*, **143**(2), pp 135-141 (1996).

10. Yu, X.-H. and Chen, G.-A. "Efficient estimation of dynamically optimal learning rate and momentum for backpropagation learning", *Proc. of IEEE ICNN-95*, pp 385-388 (1995).

11. Narendra, K.S. and Thathachar, M.A.L., *Learning Automata: An Introduction*, Prentice-hall, Englewood Cliffs (1981).

12. Menhaj, M.B. and Meybodi, M.R. "A novel learning scheme for feedforward neural nets", *Proc. of ICEE-95*, University of Science and Technology, Tehran, Iran (1995).

13. Beigy, H., Meybodi, M.R. and Menhaj, M.B. "Adaptation of learning rate in backpropagation algorithm using fixed structure learning automata", *Proc. of ICEE-98*, **III**, pp 117-123 (1998).

14. Baba, N. and Handa, H. "Utilization of hierarchical structure stochastic automata for the backpropagation method with momentum", *Proc. of IEEE ICNN-95*, pp 389-393 (1995).

15. Meybodi, M.R. and Lakshmivarhan, S. "On a class of learning algorithms which have a symmetric behavior under success and failure", *Springer Verlag Lecture Notes in Statistics*, pp 145- 155 (1984).

16. Oommen, B.J. and Croix, E.V. de St. "Graph partitioning using learning automata", *IEEE Trans. on Computers*, **45**(2), pp 195-208 (1996).

17. Beigy, H. and Meybodi, M.R. "Solving the graph isomorphism problem using learning automata", in *Proc. of 5th Annual International Computer Society of Iran Computer Conference, CISCC-2000*, Tehran Iran, pp 402-415 (Jan. 2000).

18. Beigy, H. and Meybodi, M.R. "An algorithm based on learning automata for determining the minimum number of hidden units in three layers neural networks", *Amir Kabir Journal of Science and Technology*, **12**(46), pp 111-136 (2001).

19. Meybodi, M.R. and Beigy, H. "Neural network engineering using learning automata: Determination of desired size for three layer feedforward neural network", *Journal of Faculty of Engineering*, **34**, pp 1-26 (2001)

20. Meybodi, M.R., Beigy, H. and Taherkhani, M. "Cellular learning automata", in *Proc. of 6th Annual International Computer Society of Iran Computer Conference CSICC-2001*, Isfahan, Iran, pp 153-163 (Feb. 2001)

21. Meybodi, M.R. and Lakshmivarhan, S. "A learning approach to priority assignment in two class M/M/1 queuing system with unknown parameters", *Proc. Third Yale Workshop on Applications of Adaptive Systems Theory*, Yale University, pp 106-109 (1983).

22. Srikantakumar, P.R. and Narendra, K.S. "A learning model for routing in telephone networks", *SIAM J. of Control and Optimization*, **20**, pp 34-57 (1982).

23. Barto, A.G. and Anandan, P. "Pattern-recognizing stochastic learning automata", *IEEE Trans. on Systems, Man and Cybernetics*, **SMC-15**, pp 360-375 (1985).

24. Mars, P., Chen, J.R. and Nambiar, R., *Learning Algorithms: Theory and Application in Signal Processing, Control and Communications*, CRC press, New York (1998).

25. Lakshmivarahan, S., *Learning Algorithms: Theory and Applications*, New York, Springer-Verlag (1981).

26. Meybodi, M.R. and Lakshmivarhan, S. "Optimality of a general class of learning algorithm", *Information Science*, **28**, pp 1-20 (1982).

27. Meybodi, M.R. "Results on strongly absolutely expedient learning automata", *Proc. of OU Inference Conf.*, **86**, D.R. Mootes and R. Butrick, Eds., Athens, Ohio, Ohio University Press, pp 197-204 (1987).

28. Narendra, K.S. and Lakshmivarahan, S. "Learning automata - A critique", *J. of Cybernetics and Information Science*, **1**, pp 53-65 (1977).

29. Gori, M. and Tesi, A. "On the problem of local minima in backpropagation", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **PAMI-14**, pp 76-86 (1992).

30. Frasconi, P., Gori, M. and Tesi, A. "Success and failures of backpropagation: A theoretical investigation", Technical Report, Dipartimento di Sistemi e Information, Universita di Firenze, Firenze, Italy (1992).