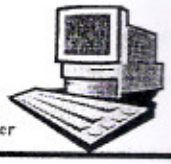
The 7th Iranian Conference on Electrical Engineering
Iran Telecommunicatior Research Center
May 17-19 1999



New Learning Automata Based Algorithms for Adaptation of Backpropagation Algorithm Parameters*

M. R. Meybodi **Hamid Beigy**

Computer Engineering Department
Amirkabir University of Technology
Tehran, Iran

Abstract

One popular learning algorithm for feedforward neural networks is the backpropagation (BP) algorithm which includes parameters, learning rate (η), momentum factor (α) and steepness parameter (λ). The appropriate selections of these parameters have large effect on the convergence of the algorithm. Many techniques that adaptively adjust these parameters have been developed to increase speed of convergence. In this paper, we shall present several classes of learning automata based solutions to the problem of adaptation of BP algorithm parameters. By interconnection of learning automata to the feedforward neural networks, we use learning automata scheme for adjusting the parameters η , α, and λ based on the observation of random response of the neural networks. One of the important aspects of proposed schemes is its ability to escape from local minima with high possibility during the training period. The feasibility of proposed methods are shown through the simulations on several problems.

Keywords: Neural Network, Backpropagation, Fixed Structure Learning Automata, Learning Rate, Steepness parameter.

1. Introduction

Backpropagation algorithm is a systematic method for training multilayer neural networks. Despite the many successful applications of backpropagation, it has many drawbacks. For complex problems it may require a long time to train the networks, and it may not train at all. Long training time can be the result of the non-optimum values for the parameters of the training algorithm. It is not easy to choose appropriate values for these parameters for a particular problem. The parameters are usually determined by trial and error and using the past experiences. For example, if the learning rate is too small, convergence can be very slow, if too large, paralysis and continuos instability can result. Moreover the best value at the beginning of training may not be so good later. Thus several researches have suggested algorithms for automatically adjusting the parameters of training algorithm as training proceeds.

Arabshahi et al. [2] proposed an error back-propagation algorithm in which the learning-rate is adapted. In this algorithm learning-rate is a function of error and changes in the error. They proposed that the learning-rate to be adjusted using a fuzzy logic control system, in which the error and changes in error are the inputs and changes in learning-rate is the output of fuzzy logic controller. Kandil et al. [3] used optimum, time-

* Part of this paper was presented at the 6th European Conf. on Intelligent Techniques and Soft Computing, Achen , Germany , 1998.

varying learning-rate for multi-layer neural network by linearizing the neural network around weight vector at each iteration. Parlos et al. [4] proposed an accelerated learning algorithm for supervised training of multi-layer neural networks named adaptive error back-propagation algorithm. In their proposed algorithm the learning-rate is a function of the error and the error gradient. Cater [5], Franzini [6], Vosl et al. [7], Tesnuro and Janssens [8], Deros and Orban [22], Darken and Moody [9], Solmon [9], Tolleraere [23], Fallside and Chan [9], Jacobs [20], and Riedmiller and Heinrich [10] has proposed other schemes for adaptation of learning rate. Sperduti and Starita [11] proposed an error back-propagation algorithm in which the steepness parameter is adapted using gradient descent algorithm. Several learning automata (LA) based procedures have been also developed [12-17]. In these methods variable structure learning automata (VSLA) or fixed structure learning automata (FSLA) have been used to find the appropriate values of parameters for the BP training algorithm. In these schemes either a separate learning automata is associated to each layer of the network or a single automata is associated to the whole network to adapt the appropriate parameters. It is shown that the learning rate adapted in such a way not only increases the rate of convergence of the network but it bypasses the local minimum in most cases.

In this paper, we propose two new classes of LA based schemes for adaptation of appropriate learning rate or steepness parameters for BP algorithms. Unlike the existing LA based schemes, in these schemes one learning automata is assigned to every link or every neuron in the network for determining the parameters for that link or neuron. The simulation results show the feasibility of the proposed method and its superiority to the existing LA based schemes. The proposed schemes have two important aspects: higher speed of convergence and a high probability of escaping from the local minima. In order to evaluate the performance of proposed schemes simulations are carried out on four learning problems: digit recognition, encoding, odd parity, and symmetry problems problem and the results are compared with results obtained from standard BP. These problems are chosen because they posses different error surfaces and collectively present an environment that is suitable to determine the effect of proposed method.

8 × 8 Dot Numeric Font Learning: There are numbers 0, ... , 9, and each represented by a 8 × 8 grid of black and white dot as shown in figure 1. The network must learn to distinguish these numbers. The network architecture used for this problem consists of 64 input units which are connected to 6 hidden units which are connected to 10 output units.

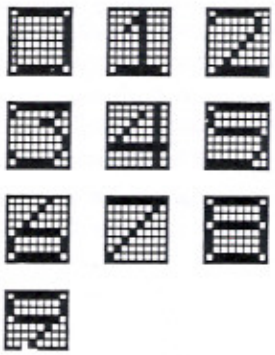


Figure 1

Odd-Parity Problem: In this problem a string of N input is applied to the network, the output of network is zero (one) if number of ones in the input is odd (even) [1]. This network has N input units, N hidden units, and one output units.

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 [1]. The network architecture used for solving the problem consist of 8 input units, 3 hidden units, and 8 output unit.

Symmetry Problem: This problem classifies input string as to whether or not they are symmetric about center[1]. The network architecture used for solving this problem consists of 8 input units, 2 hidden units, and 1 output unit.

The rest of the paper is organized as follows: Section 2 briefly presents the basic backpropagation algorithm and learning automata. Application of learning automata for adaptation of learning rate, momentum factor, and steepness parameter is given in section 3I. Section 4 presents the proposed learning automata based schemes. The simulation results are given in section 5. Section 6 concludes the paper.

2. Backpropagation Algorithm and Learning Automata

In this section, in all brevity, we discuss the fundamentals of backpropagation learning algorithm and learning automata.

Backpropagation Algorithm: Error backpropagation training algorithm which is an iterative gradient descent algorithm is a simple way to train multilayer feedforward neural networks [5]. The BP algorithm is based on the gradient descent rule:

$$W(n+1) = W(n) + \eta G(n) + \alpha[W(n) - W(n-1)] \quad (1)$$

where W is the weight vector, n is the iteration number, η is learning rate, α is momentum factor, and G is gradient of error function that is given by:

$$G(n) = -\nabla E(n) \quad (2)$$

Where E is the sum of squared error given by:

$$E(n) = \frac{1}{2} \sum_{p=1}^{\#patterns} \sum_{j=1}^{\#outputs} [T_{p,j} - O_{p,j}]^2 \quad (3)$$

Where $T_{p,j}$ and $O_{p,j}$ are desired and actual outputs for pattern p at output node j. One of the major problems encountered during implementation of the BP learning rule is proper choice and update of the learning rate η to allow convergence, while keeping the number required iterations at a reasonable number. One of main reasons for investigating the possibility of the adaptive learning rate rule is the desire to reduce the sensitivity of the learning on the learning rate, without adding more tuning parameters.

In the BP algorithm framework, each computational unit computes the same activation function. The computation of the sensitivity for each neuron requires the derivative of activation function, therefore this function must be continuos and differentiable. The activation function is normally a sigmoid function chosen between $1/1+\exp(-\lambda net)$ and $\tanh(\lambda net)$. The coefficient of the exponent of the exponential term determines the steep of linearity of that function. The steepness parameter λ is often set to a constant value and not changed by the learning algorithm. We gain much flexibility, if we move the net inputs of the sigmoidal functions near to their active regions, where the associated gradient are not very close to zero. This makes the BP algorithm not be trapped to 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 better to center each sigmoid to be inside the active region of the sigmoidal function.

The momentum term in weight adaptation equation (1) causes large change in the weight if the changes are currently large, and will decrease as the changes become less. This means that the network is less likely to get stuck in local minima early on, since the momentum term will push the changes over local downward trend. Momentum is of great assistance in speeding up convergence along shallow gradients, allowing 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 which finish at minima. Convergence along these ravines is slow, and usually the algorithm oscillates across the ravine valley as it moves towards a solution. This is difficult to speed up without increasing the chance of overshooting the minima, but the addition of the momentum term is fairly successful. This difficulty could be removed if we select the momentum factor to be small at the near of minima and to be large far from minima.

Learning Automata: Learning automata (LA) can be classified into two main families, fixed and variable structure learning automata [18][26-28]. Examples of the FSLA type that we use in this paper are Tsetline, Krinsky, TsetlineG, and Krylov automata. A fixed structure learning automaton is quintuple $\langle \underline{\alpha}, \underline{\phi}, \underline{\beta}, F, G \rangle$ where:

1) $\alpha = (\alpha_1, \ldots, \alpha_R)$ is the set of actions that it must choose from.
2) $\Phi = (\Phi_1, \ldots, \Phi_s)$ is the set of states.
3) $\beta = \{0, 1\}$ is the set of inputs where "1" represents a penalty and "0" a reward.
4) $F : \Phi \times \beta \rightarrow \Phi$ is a map called the transition map. It defines the transition of the state of the automaton on receiving an input, F may be stochastic.
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 the time n. $\beta(n)$ is an element of $\beta=\{0, 1\}$ and is the feedback response of the environment to the automaton. The environment penalize (i.e., $\beta(n) = 1$) the automaton with the penalty c_i, which is the action dependent. On

the basis of the response $\beta(n)$, the state of the automaton $\Phi(n)$ is updated and a new action chosen at the time $(n+1)$. Note that the $\{c_i\}$ are unknown initially and it is desired that as a result of interaction with the environment the automaton arrives at the action which presents it with the minimum penalty response in an expected sense. If the probability of the transition from one state to another state and probabilities of correspondence of action and state are fixed, the automaton is said fixed-structure automata and otherwise the automaton is said variable-structure automata. We summarize some of fixed-structure learning automaton and variable structure automaton in the following paragraphs.

The two-state automata ($L_{2,2}$) : This automata has two states, ϕ_1 and ϕ_2 and two actions α_1 and α_2. The automata accepts input from a set of {0, 1} and switches its states upon encountering an input 1 (unfavorable response) and remains in the same state on receiving an input 0 (favorable response). An automata that uses this strategy is refereed as $L_{2,2}$ where the first subscript refers to the number of states and second subscript to the number of actions.

The two-action automata with memory ($L_{2N,2}$) : This automata 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. While the automata $L_{2,2}$ switches from one action to another on receiving a failure response from environment, $L_{2N,2}$ keeps an account of the number of success and failures received for each action. It is only when the number of failures exceeds the number of successes, or some maximum value N, the automata switches from one action to the another. The procedure described above is one convenient method of keeping track of performance of the actions α_1 and α_2. As such, N is called memory depth associated with each action, and automata is said to have a total memory of 2N. For every favorable response, the state of automata moves deeper into the memory of corresponding action, and for an unfavorable response, moves out of it. This automata can be extended to multiple action automata and this automata is named $L_{2N,2}$ automata. The state transition graph of $L_{2N,2}$ automata is shown in figure 2.

The Krinsky automata :This automata behaves exactly like $L_{2N,2}$ automata when the response of the environment is unfavorable, but for favorable response, any state ϕ_i (for i = 1, . . . , N) passes to the state ϕ_1 and any state ϕ_i (for i = N+1, ... , 2N) passes to the state ϕ_{N+1}. This implies that a string of N consecutive unfavorable responses are needed to change from one action to another. The state transition graph of Krinsky automata is shown in figure 3.

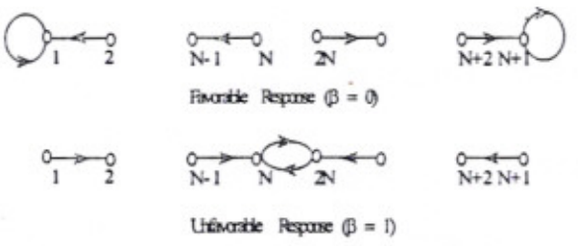


Figure 2: The state transition graph for $L_{2N,2}$

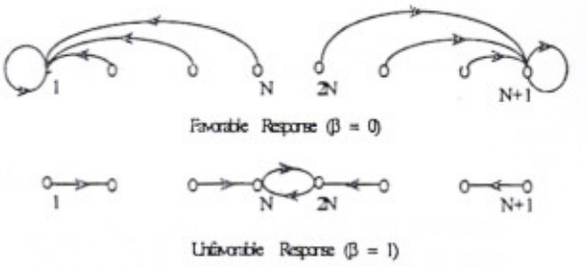


Figure 3: The state transition graph for Krinsky Automata

The Krylov automata : This automata has state transition that are identical to the $L_{2N,2}$ automata when the output of the environment is favorable. However, when the response of the environment is unfavorable, a state ϕi (i ≠1, N, N+1, 2N) passes to a state ϕ_{i+1} with probability 0.5 and to a state ϕ_{i-1} with probability 0.5. When i =1 or i =N+1, ϕ_i stays in the same state with probability 0.5 and moves to ϕ_{i+1} with the same probability. When i = N, automata state moves ϕ_{N-1} to ϕ_{2N} and with the same probability 0.5. When i = 2N, automata state moves ϕ_{2N-1} to ϕ_N with the same probability 0.5. The state transition graph of Krylov automata is shown in figure 4.

In this paper we refer to an automata by the name of automata followed by the list of parameters for that automata, the first parameter refers to the number of actions and the second parameter refers to the depth for each action.

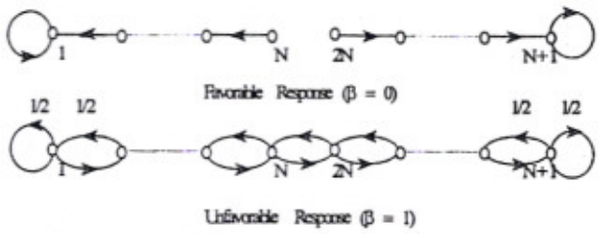


Figure 4: The state transition graph for Krylov Automata

Variable structure Automata: Variable-structure automata is represented by sextuple $\langle \underline{\beta}, \underline{\phi}, \underline{\alpha}, \underline{P}, G, T \rangle$, where $\underline{\beta}$ a set of inputs actions, $\underline{\phi}$ is a set of internal states, $\underline{\alpha}$ 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 learning algorithm for updating the action probabilities. Various learning algorithms have been reported in the literature [18]. 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

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

if β is zero and P is unchanged if β is one. The parameter θ is called step length, it determines the amount of increases (decreases) of the action probabilities. In linear reward-penalty algorithm (L_{R-P}) scheme the recurrence equation for updating p is defined as

$$p_j(k) = \begin{cases} p_j(k) + \theta\left(1 - p_j(k)\right) & \text{if } i = j \\ p_j(k) + \theta\ p_j(k) & \text{if } i \neq j \end{cases} \quad \beta = 0 \qquad p_j(k) = \begin{cases} p_j(k)\left(1 - \gamma\right) & \text{if } i \neq j \\ p_j(k) + \gamma\left(1 - p_j(k)\right) & \text{if } i = j \end{cases} \quad \beta = 1$$

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

3. LA Based Schemes For Adaptation of BP Parameters

In this section, we first, briefly describe previous LA based schemes [12-17] for adaptation of BP parameters and then introduce two new classes of LA based schemes. In all of the existing schemes, one or more automatas have been associated to the network. The learning automata (or automatas) based on the observation of the random response of the neural network, adapt one or more of BP parameters. The interconnection of learning automata and neural network is shown in figure 5. Note that the neural network is the environment for the learning automata. The learning automata according to the amount of the error received from neural network adjusts the parameters of the BP algorithm. The actions of the automata correspond to the values of the parameters being calculated and input to the automata is some function of the error in the output of neural network.

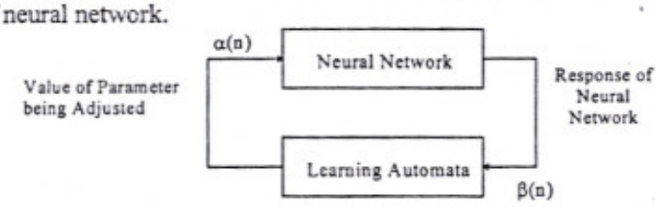


Figure 5: The interconnection of learning automata and neural network

A function of error between the desired and actual outputs of network is considered as the response of environment. A window on the past values of the errors are swiped and the average value of the error in this window computed. If the difference of the average value in the two last steps is less than the predefined

threshold value, the response of the environment is favorable and if this difference of average value is greater than the threshold value, the response of the environment is unfavorable.

Existing LA based procedures for adaptation of BP parameters can be classified into two groups which we call them group *A* and group *B*. In group *A* schemes, an automaton is used for the whole network [12,14] whereas in group *B* schemes, separate automatas one for each layer (hidden and output) are used [13, 15, 16, 17]. Each group *A* and *B* depending on the type of automata used (fixed or variable structure) can be classified into two subgroups. The parameter adapted by group *A* schemes will be used by all the links or neurons of the networks and therefore these schemes fall in the category of global parameter adaptation method, whereas group *B* schemes by adapting the parameter for each layer independently may be referred to as quasi-global parameter adaptation methods. For the sake of convenience in presentation, we use the following naming conventions to refer to different LA based schemes in class *A* and class *B*.

Automata-AV(γ): A scheme in class *A* for adjusting parameter γ which uses variable structure learning automata **Automata**.

Automata-AF(γ): A scheme in class *A* for adjusting parameter γ which uses fixed structure learning automata **Automata**.

Automata$_1$-Automata$_2$-BV(γ): A scheme in class *B* for adjusting parameter γ which uses variable structure learning automata **Automata$_1$** for hidden layer and variable structure learning automata **Automata$_2$** for output layer.

Automata$_1$-Automata$_2$-BF(γ): A scheme in class *B* for adjusting parameter γ which uses fixed structure learning automata **Automata$_1$** for hidden layer and fixed structure learning automata **Automata$_2$** for output layer.

The letters **F** and **V** in above names denotes FSLA and VSLA , respectively. For all the LA based schemes reported, it is shown through simulation that the use of LA for adaptation of BP learning algorithm parameters increases the rate of convergence by a large amount. Figure 6 borrowed from [16], compares the effectiveness of different LA based schemes in class *A* for adaptation of learning rate for the 8×8 dot numeric font recognition problem. In this simulation the threshold of 0.01 and window size of 1 is chosen. For linear reward-penalty automata the reward and penalty coefficient 0.001 and 0.0001 are chosen. It is reported that FSLA based schemes have performance much higher than the VSLA based schemes [15]. Also simulation studies have shown that by using LA based scheme for adaptation of learning rate or momentum factor we can compute a new point that is closer to the optimum than the point computed by BP algorithm which uses a fixed predetermined learning rate or momentum factor[15, 16].

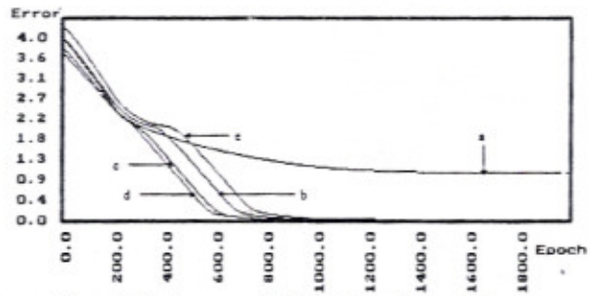


Figure 6: Performance of different class A based schemes

a: Standard BP b: Tsetline(4,4)-AF (η) c: Krinskv(2,4)-AF (η) d: Krylov(2,4)-AF (η) e: L_{RP}-AV (η)

4. New classes of schemes for adaptation of BP parameters

In this section we propose two new classes of LA based schemes called class *C* and *D*. In a class *C* scheme one automata is associated to each link of the network to adjust the parameter for that link and in a class *D* scheme one automata is associated to each neuron of the network to adjust the parameter for that neuron. Group *C* and *D* schemes may be referred to as the local parameter adaptation methods. We use **Automata-CV(γ)** and **Automata-CF(γ)** to refer to the schemes in class *C* and **Automata-DV(γ)** and **Automata-DF(γ)** to refer to the schemes in class *D*. The letters **F** and **V** in above names denote FSLA and VSLA, respectively.

We use class *C* schemes for adaptation of learning rate and class *D* schemes for adaptation of steepness parameter. In class *C* and *D* schemes, the automata receives favorable response from the environment if the algebraic sign of derivative in two consecutive iterations is the same and receives unfavorable response if the algebraic sign of the derivative in two consecutive iterations alternates. The following algorithms describe class *C* and class *D* schemes.

In the following algorithms to compute the partial derivative of error with respect to steepness parameter ($\frac{\partial E}{\partial \lambda}$), we minimize the error given by equation (3). Assume that net_I^L represents the net output of I^{th} neuron in L^{th} layer, which is given by

$$net_I^L = \sum_{k=0}^{N_{L-1}} W_{K,I}^L \times O_K^{L-1}$$

O_K^L is output of k^{th} neuron in L^{th} layer given by $O_K^L = f_K^L(net_K^L)$, where f_K^L shows the activation function of k^{th} neuron in L^{th} layer which is one of sigmoidal or tanh functions. Differentiation of E with respect λ_K^L yields to:

$$\frac{\partial E}{\partial \lambda_K^L} = -\delta_K^L \times \frac{\partial f}{\partial \lambda_K^L} \qquad \text{where} \qquad \delta_K^L = \begin{cases} (T_K - O_K) & \text{if L is output layer} \\ \sum_{J=1}^{N_{L+1}} \delta_J^{L+1} \times W_{K,J}^{L+1} \times \frac{\partial f}{\partial net_J^{L+1}} & \text{if L is hidden layer} \end{cases}$$

```
procedure C_Scheme_BP (Automata)
  Initialize the weights to small random values.
  initialize the parameters for automaton Automata.
  repeat
    for all training patterns (X, T) in the training set do
      Call FeedForward
      Call ComputeGradient
      for all layers in the network do
        for all nodes in I^th layer do
          for all weights w for n^th node in I^th layer do
            if Sign(∂E/∂w, k) = Sign(∂E/∂w, k - 1) Then
              //The sign at iteration k and k-1 is the same
                η = Call Automata (0)      // β is 0
              else
                η = Call Automata (1)      // β is 1
              End if
            End for
          End for
        End for
      End for
      Call UpdateWeights
      // Batch weights updating is used
  until k > N.  // N is maximum training epoch
End procedure
```

Figure 7: BP with a class *C* scheme

```
procedure D_Scheme_BP (Automata)
  Initialize the weights to small random values.
  initialize the parameters for the automaton Automata.
  repeat
    for all training patterns (X, T) in the training set do
      Call FeedForward
      Call ComputeGradient
      for all layers in the network do
        for all steepness parameters β in I^th layer do
          if Sign(∂E/∂λ, k) = Sign(∂E/∂λ, k - 1) Then
          //The sign at iteration k and k-1 is the same
              λ = Call Automata (0)        // β is 0
          else
              λ = Call Automata (1)        // β is 1
          End if
        End for
      End for
    End for
    Call UpdateWeights
    // Batch weights updating is used
  until k > N  // N is maximum training epoch
End procedure
```

Figure 8: BP with a class *D* scheme

A deterministic fixed structure learning automaton: In what follows, we introduce a new fixed structure learning automata called J-automata. This automata can be used by class *C* and class *D* schemes to achieve higher degree of performance than the other known schemes in class *C* or class *D*. The proposed automata which we denote it by $J_{KN,K}$ has KN states and K actions and attempts to incorporate the past behavior of the system in its decision rule for choosing the sequence of actions. States with numbers (k-1)N+1 through kN correspond to action k. The state transition graph of this automata for favorable response and unfavorable response is shown in figure 9.

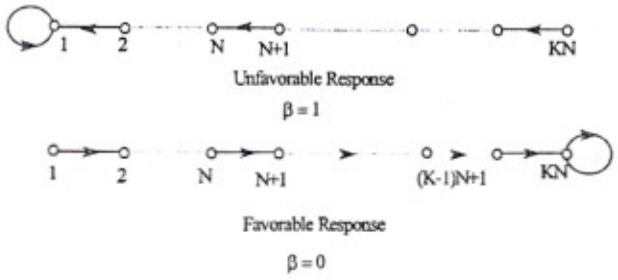


Figure 9: The state transition graph for $J_{KN,K}$

The favorable response is produced by the environment if the algebraic sign of the derivative in two consecutive iterations is the same and unfavorable response produced by the environment if the algebraic sign of the derivative in two consecutive iterations alternates. When this automata is used to adjust the learning rate (steepness parameter), each action correspond to one of the values of the learning rate (steepness parameter). The automata, on reward move to the higher number state (hoping eventually increases the value of the parameter) and on penalty move to the lower number state (hoping eventually decreases the value of the parameter).

It is only when the sign of derivative remains the same for N consecutive iterations or alternates for N consecutive iteration the automata changes its action (switches from one value for the related parameter to another value). Therefore N is memory depth associated with each action and automata is said to have a total memory of KN. If parameter μ can assume K values, $\mu_1 \le \mu_2 \le \ldots \le \mu_K$ then states (k-1)N + 1 through kN correspond to the value of μ_k. Clearly such an assignment of values of parameter μ to the states of the automata causes the value used by BP increases if in N consecutive iterations the sign of the derivative do not change and decreases if in N consecutive iterations the sign of derivative alternates.

Remark 1: Variable learning rate scheme (VLR) In this remark we first explain the Variable learning rate scheme (VLR) scheme and then explain why J_CF (η) performs better than VLR scheme. Variable learning rate is a scheme in which the learning rate is varied according to the performance of the algorithm [19]. If the error decreases after a weight update, then the learning rate is increased by some factor (e.g., 1.05). If the error increases more than some set of percentage (typically one to five percent), then the weight update is discarded and the learning rate is decreased by some factor (e.g., 0.7) and the momentum term (if it is used) is set to zero. When a successful step is taken the momentum term is reset to its original value. If the algorithm is working well, and the error continuos to go down, then learning rate will increase and convergence will speedup.

Assume that the network has n adjustable parameters $\pi_1, \ldots, \pi_n$. Gradient of error function with respect to $\pi_1, \ldots, \pi_n$ is defined as:

$$\nabla E = \left[\frac{\partial E}{\partial \pi_1}, \frac{\partial E}{\partial \pi_2}, \ldots, \frac{\partial E}{\partial \pi_n}, \right]^T$$

Gradient vector ∇E extends in the direction of the greatest rate of change of E but it doesn't mean that all $\nabla_k E$ (the k^{th} element of ∇E) have the same algebraic sign. In other word the greatest rate of increase(decrease) of E doesn't imply the same sign for projections of $\nabla_k E$ along all axis.

VLR scheme decreases (increases) learning rate as the result of increase (decrease) of E based on gradient. In VLR scheme a single learning rate will be adapted and be used by BP to walk along all directions, which may create oscillation along some axis and leads to lower rate of convergence. On the contrary, in **C** (***D***) schemes, since each link (neuron) has its own learning rate (steepness) (that is, the learning rate (steepness parameter) for every axis is adapted independently), oscillation may be decreased and as a result higher rate of convergence can be obtained.

Simultaneous adaptation of learning rate and steepness parameters: The rate of convergence can be improved if both learning rate and steepness parameter are adapted simultaneously. The following algorithm describes the simultaneous use of class ***C*** and class ***D*** schemes for adaptation of learning rate and steepness parameters. In this algorithm, a class ***C*** scheme is used for adaptation of learning rate and a class ***D*** scheme is used for adaptation of steepness parameter. A scheme that simultaneously adapts learning rate and steepness

parameter is denoted by **Automata1-Automata2-CDF(μ,λ)**, if FSLA is used and **Automata1-Automata2-CDV(μ, λ)**, if VSLA is used.

Simultaneous adaptation of learning rate and momentum factor: A simple method of increasing the learning rate and stability of training algorithm is to modify the standard BP by including the momentum factor [1] as given in equation (1). The changes in the weight at iteration of n is given by:

$$\Delta W(n) = \eta G(n) + \alpha \Delta W(n-1) \tag{4}$$

Solving the difference equation (4) gives to the following time series equation.

$$\Delta W(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} G(t) \tag{5}$$

By inspection of equation (5), we may make the following useful observations:

1. The current adjustment of $\Delta W(n)$ represents the sum of an exponentially weighted time series. This equation converged if and only if $0 \le |\alpha| < 1$.
2. When G has same algebraic sign on consecutive iterations, $|\Delta W|$ grows, and W is adjusted by a large amount. Hence the inclusion of momentum term accelerate the convergence of algorithm. To accelerate more, the momentum factor must have large value as much as possible.
3. When the algebraic sign of G alternates in consecutive iterations, the $|\Delta W|$ shrinks and W is adjusted by a small amount. Hence the inclusion of momentum term stabilize the convergence of algorithm. To accelerate more, the momentum factor must have small value as much as possible.

From the above observation, we may conclude that the momentum factor could be adjusted by $J_{KN,K}$ automata in the same manner as the learning rate. The following algorithm describes the simultaneous adaptation of learning rate and momentum factor. This scheme is denoted by **Automata1-Automata2-CF**(η, α).

```
procedure Simultaneous_C_D_BP (Automata1, Automata2)
  Initialize the weights to small random values.
  initialize the parameters for the Automata1 & Automata2.
  repeat
    for all training patterns (X, T) in the training set do
      Call FeedForward
      Call ComputeGradient
      for all layers in the network do
        for all nodes in l^th layer do
          for all weights w for n^th node in l^th layer do
            if Sign (∂E/∂w, k) = Sign (∂E/∂w, k - 1) Then
            //The sign at iteration k and k-1 is the same
                η = Call Automata1 (0)      // β is 0
            else
                η = Call Automata1 (1)      // β is 1
              End if
            End for
          End for
          for all steepness parameters λ in l^th layer do
            if Sign (∂E/∂λ, k) = Sign (∂E/∂λ, k - 1) Then
              //The sign at iteration k and k-1 is the same
              λ = Call Automata2 (0)      // β is 0
            else
              λ = Call Automata2 (1)      // β is 1
            End if
          End for
        End for
      End for
      Call UpdateWeights
      // Batch weights updating is used
    until k > N.  // N is maximum training epoch
End procedure
```

Figure 10: *Automata1- Automata2-CDF(η, λ)* scheme

```
procedure Simultaneous_C_Scheme_BP (Automata1, Automata2)
  Initialize the weights to small random values.
  Initialize the parameters for automatas Automata1 & Automata2.
  repeat
    for all training patterns (X, T) in the training set do
      Call FeedForward
      Call ComputeGradient
      for all layers in the network do
        for all nodes in l^th layer do
          for all weights w for n^th node in l^th layer do
            if Sign (∂E/∂w, k) = Sign (∂E/∂w, k - 1) Then
              //The sign at iteration k and k-1 is the same
                η = Call Automata1 (0)      // β is 0
                α = Call Automata2 (0)      // β is 0
              else
                η = Call Automata1 (1)      // β is 1
                α = Call Automata2 (1)      // β is 1
              End if
            End for
          End for
        End for
        Call UpdateWeights
        // Batch weights updating is used
    until k > N.  // N is maximum training epoch
End procedure
```

Figure 11: **Automata1-Automata2**-CF(η, α) scheme

5. Simulations

Typical simulations for these four problems for different parameter adaptation schemes are shown in figures 12 through 17. Figure 12 compares the performance of different class *A* schemes with J_CF scheme. Figure 13 indicates that J_CF scheme has higher speed of convergence than any known scheme in class *C*. Figure 14 compare the J_CF scheme with the best scheme in class *B*. Figures 15 through 17 indicate that if μ and λ adapted simultaneously, the performance of the BP algorithm increases by a large amount. Figures 18 through 20 compare the performance of J_J_CDF (μ, λ) with VLR scheme and schemes proposed by Arabshahi (Fuzzy BP) and Darken and Moody (SC). For all the simulations, we have taken the momentum factor (α) to be zero. The plot for each simulation is averaged over 200 runs.

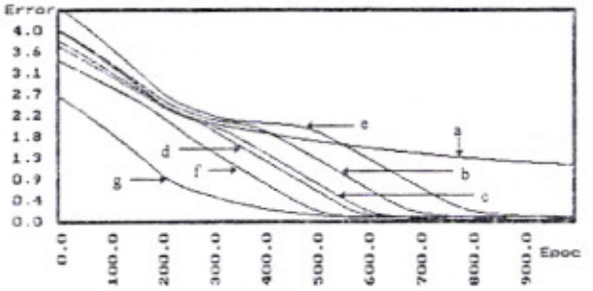


Figure 12: Digit problem
a: standard BP b: Tsetline(4, 4)-AF (η)
c: Krinsky(2, 4)-AF (η) d: Krylov(2, 4)-AF (η)
e: $L_{R\text{-}P}$-AV (η) f: VLR g: J(2, 1)-CF (η)

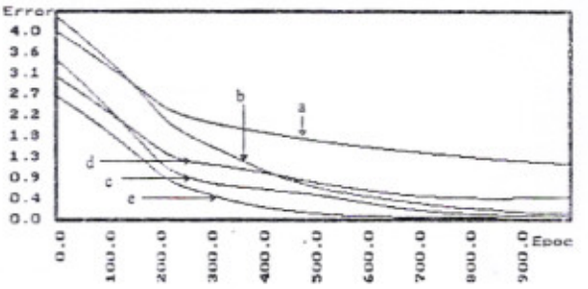


Figure 13: Digit problem
a: standard BP
b: Tsetline(4, 4)-CF (η) c: Krinsky(2, 4)-CF (η)
d: Krylov(2, 4)-CF (η) e: J(2, 1)-CF (η)

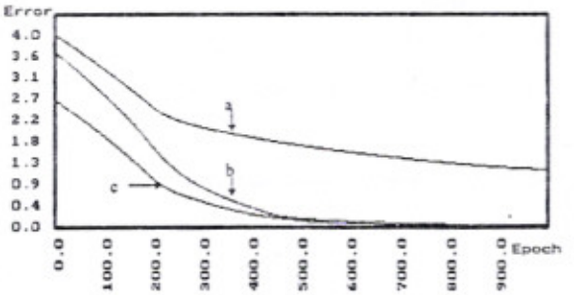


Figure 14: Digit problem
a: Standard BP
b: Tsetline(4,6)-Tsetline(2,4)-BF(η) c: J-CF(η) (10, 1)

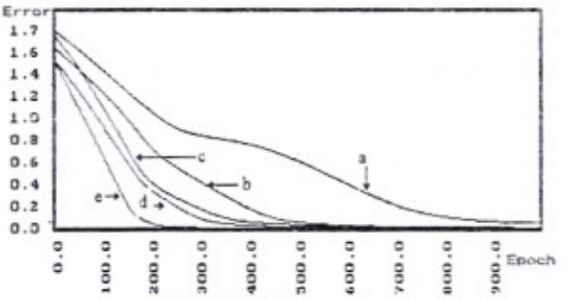


Figure 15: Parity problem
a: Standard BP b: VLR c:J(2, 1)-CF (η)
d: J(2, 1)-CF (λ) e: J(2, 1)-J(2, 1)-CDF (η, λ)

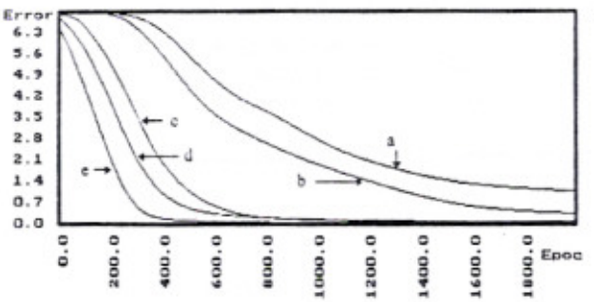


Figure 16: Encoding problem
a: Standard BP b: VLR c:J(2, 1)-CF (η)
d: J(5, 6)-CF (λ) e: J(2, 1)-J(5, 6)-CDF (η, λ)

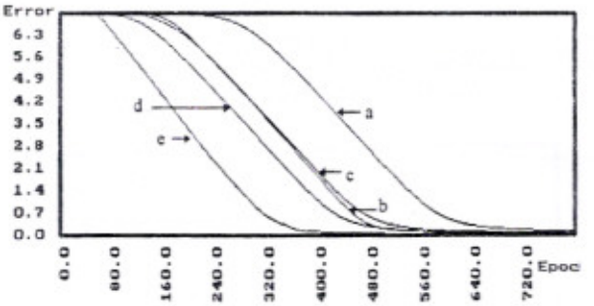


Figure 17: Symmetry problem
a: Standard BP b: VLR c:J(5, 6)-CF (λ)
d: J(2, 1)-CF (η) e: J(2, 1)-J(5, 6)-CDF (η, λ)

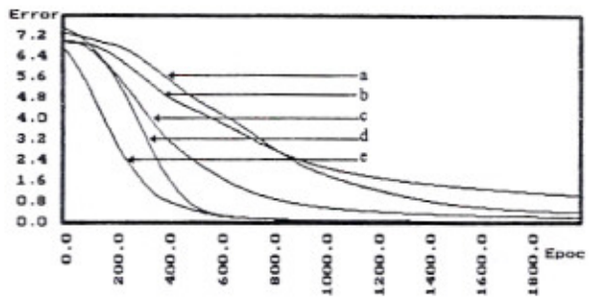


Figure 18: Encoding Problem
a: SC Scheme b: Standard BP c:VLR Scheme
d: Fuzzy BP e: J(2, 1)-J(5, 6)-CDF (μ, λ)

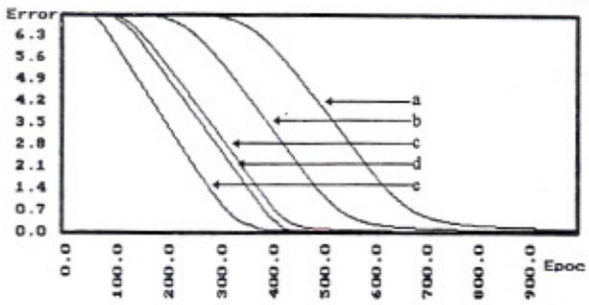


Figure 19: Symmetry Problem
a: SC Scheme b: Standard BP c:VLR Scheme
d: Fuzzy BP e: J(2, 1)-J(5, 6)-CDF (μ, λ)

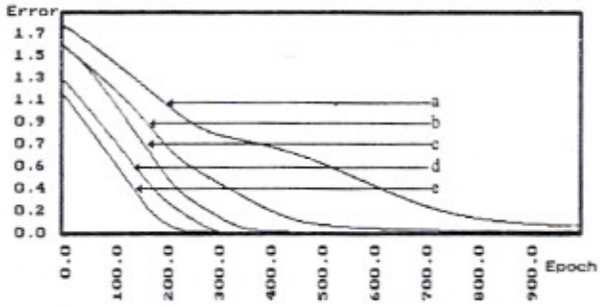


Figure 20: Parity Problem
a: SC Scheme b: Standard BP c:VLR Scheme
d: Fuzzy BP e: J(2, 1)-J(2, 1)-CDF (μ, λ)

Figures 21 through 24 show the performance of different schemes when both learning rate and momentum factor adapted.

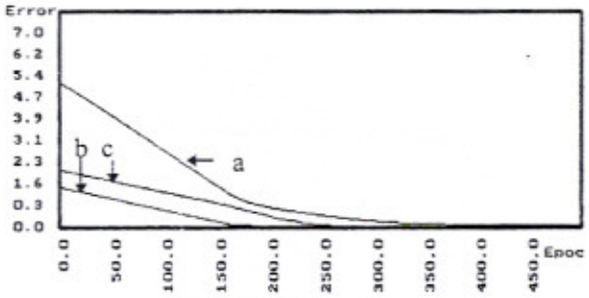


Figure 21 : Digit problem
a: J(5, 10)-CF(η) b: J(5, 10)-J(5, 10)-CF(η,α)
c: J(5, 10)-CF(η) with constant momentum factor

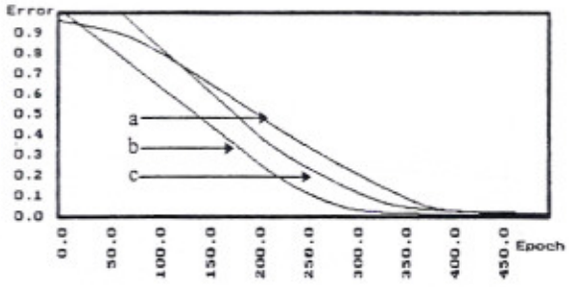


Figure 22 : Parity problem
a: J(5,10)-J(5,10)-CDF(η,λ) b: J(5, 10)-J(5, 10)-CF(η,α)
c: J(5, 10)-CF(η) with fixed momentum factor

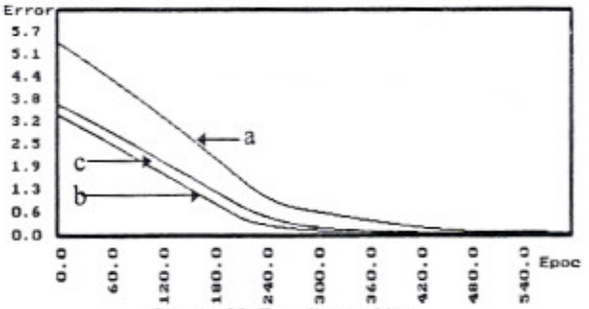


Figure 23: Encoding problem
a: J(5, 10)-J(5, 6)-CDF(η, λ) b: J(5, 10)-J(5, 10)-CF(η,α)
c: J(5, 10)-CF(η) with fixed momentum factor

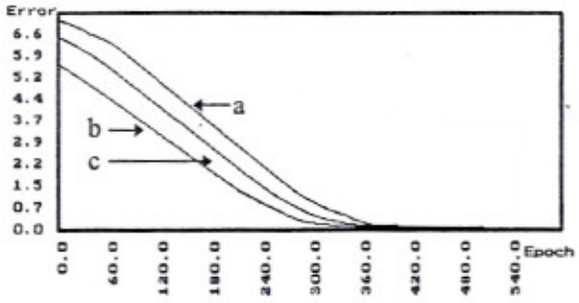


Figure 24 : Symmetry problem
a: J(15, 10)-J(5, 4)CDF(η, λ) b: J(15, 10)-J(15, 10)-CF(η,α)
c: J(15, 10)-CF(η) with fixed momentum factor

Remark 2: **J-CF(η)** scheme has a close relationship with the Jacobs's heuristics . Jacobs [20] has suggested the following heuristics as guidelines for accelerating the convergence of BP learning algorithm through learning rate adaptation.

1. Every adjustable network parameter of cost function should have its own individual learning-rate parameter.
2. Every learning-rate parameter should be allowed to vary from one iteration to the next.
3. When the derivative of cost function with respect to the synaptic weight has same algebraic sign for several consecutive iterations of the algorithm, the learning-rate parameter for that particular weight should be increased.
4. When the algebraic sign of the derivative of cost function with respect to the synaptic weight alternates for several consecutive iterations of the algorithm, the learning-rate parameter for that particular weight should be decreased.

Considering the fact that each link of the neural network has its own automata for adaptation of learning parameter, and with the definition of favorable and unfavorable response given before, and also inspecting the transition graphs for these automatas, we can see that among different schemes in class **CF** only **J-CF (η)** scheme implements all four heuristics of Jacobs. The other schemes in class **CF** such as **Krylov_CF, Krinsky_CF,** and **Tsetline_CF** implement only three of the four heuristics of Jacobs.

Remark 3: **J-CF** (η) and **J-DF** (λ) schemes become the standard BP when the memory depth for each action (N) approaches infinity. This is because of the fact that when N is very large it becomes improbable for the $J_{KN,K}$ automata to change action and as a result, a fixed value for the learning rate will be used throughout the training period. Figure 25 shows the effect of memory depth on speed of learning.

Remark 4: **J-CF** (η) and **J-DF** (λ) schemes become the standard BP when the number of actions(K) approaches infinity. This is because of the fact that when K is very large, the changes in the BP parameters is very small and for a certain amount of changes in the value of the parameter, the automata needs to make large number of states changes. This effect is the same as the effect we observed for large memory depth and small number of actions. Figure 26 shows the effect of number of actions on speed of learning.

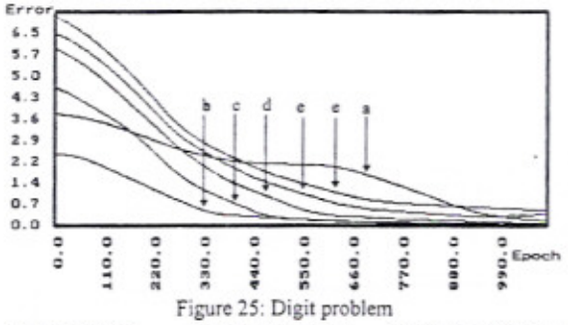


Figure 25: Digit problem

a: Standard BP b: J(10, 10)-CF (η) c: J(10, 250)-CF (η)
d: J(10, 500)-CF (η) e: J(10, 750)-CF (η) f: J(10, 1000)-CF (η)

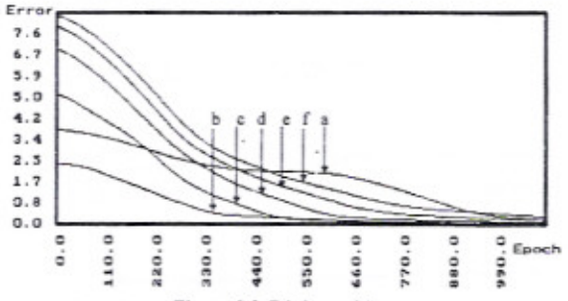


Figure 26: Digit problem

a: Standard BP b: J(10, 10)-CF (η) c: J(250, 10)-CF (η)
d: J(500, 10)-CF (η) e: J(750, 10)-CF (η) f: J(1000, 10)-CF (η)

Remark 5: J(5,2)-DF(λ) (J(5,2)-CF(η)) scheme is used for parity problem and probability of each action for a randomly selected neuron (weight) is plotted for each scheme. As shown in figures 27 and 28 the system converges to the action with the higher value. This effect has also been observed in genetic algorithms for determination of BP parameters [25]. The value of actions are 0.4, 0.8, 1.2, 1.6, and 2.

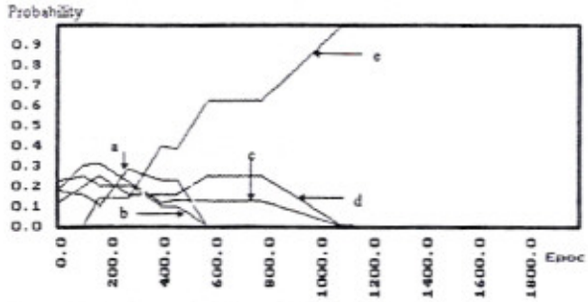


Figure 27: Action probability of J(5, 2)_DF(λ) scheme for parity problem

a: Action 1 b: Action 2 c: Action 3 d: Action 4 e: Action 5

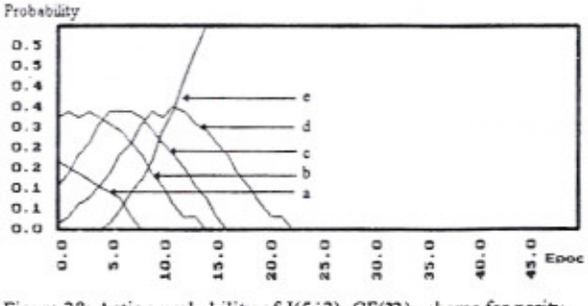


Figure 28: Action probability of J(5, 2)_CF(η) scheme for parity problem

a: Action 1 b: Action 2 c: Action 3 d: Action 4 e: Action 5

Remark 6: Self-Adaptive BP (SAB) was developed independently by Jacobs [20] and Devos and Orban [22]. SAB is a local method in which every weight has its own learning rate. In this method every learning rate on every dimension is adapted based on the error surface independently. The learning rate increased if in two consecutive iterations gradient have same sign and the learning rate should be small if sign of gradient in two consecutive iterations alternates. SAB performs better than the BP, because it can adjust the learning rate over a wide range, but it has two drawbacks: (1) The selection of initial value η is hard to determine (2) If sign of gradient alternates the learning rate is reseted to initial value. SuperSAB algorithm which is proposed by Tollenaere [23] overcomes these problems. The figures 29 and 30 describe these two algorithms in more details. In these algorithms $w_{ij}^{l}[k]$, $\eta_{ij}^{l}[k]$, and $G_{ij}^{l}[k]$ denote the weight, learning rate, and negative of the partial derivative of error with respect to the $w_{ij}^{l}[k]$ at iteration k, respectively. Algorithms for SAB and super SAB schemes are given in figures 29 and 30, respectively.

SAB

Choose an initial learning rate coefficient η

Set the learning rate for all weights $\eta_{ij}^{l}[0] = \eta$

Do

Do a BP step without momentum term

if $G_{ij}^{l}[k]$ has same sign **Then**

$\eta_{ij}^{l}[k+1] = \rho \times \eta_{ij}^{l}[k]$ // $\rho > 1$

else

$\eta_{ij}^{l}[k+1] = \eta$

Estimate a good weigth $w_{ij}^{l}[k+1]$ by interpolation

Do a number of BP steps with momentum term

End If

End Do

End SAB

Figure 29: SAB algorithm

SuperSAB

Choose an initial learning rate coefficient η

Set the learning rate for all weights $\eta_{ij}^{l}[0] = \eta$

Do

Do a number of BP steps with momentum term

if $G_{ij}^{l}[k]$ has same sign **Then**

$\eta_{ij}^{l}[k+1] = \rho \times \eta_{ij}^{l}[k]$ // $\rho > 1$

else

$\eta_{ij}^{l}[k+1] = \sigma \times \eta_{ij}^{l}[k]$ // $\sigma < 1$

Undo the previous weight update

End If

End Do

End SuperSAB

Figure 30: SuperSAB algorithm

These two schemes are simulated on parity, encoding, and numeric font recognition problems and compared with standard BP, VLR, J_CF, and J_CDF schemes. The simulation results which are given in figures 31 through 33 show the superiority of J_CF and J_CDF schemes.

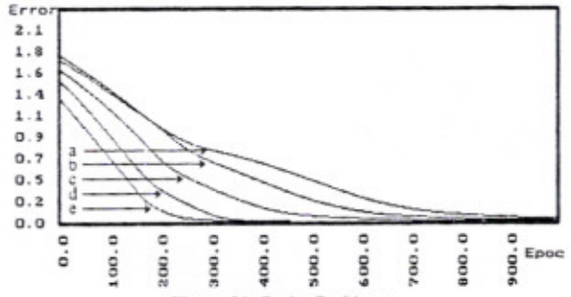


Figure 31: Parity Problem
a:Standard BP b: SAB c: Super SAB
d:J(5, 3)-CF (η) e: J(5, 3)-J(5, 3)-CDF(η, λ)

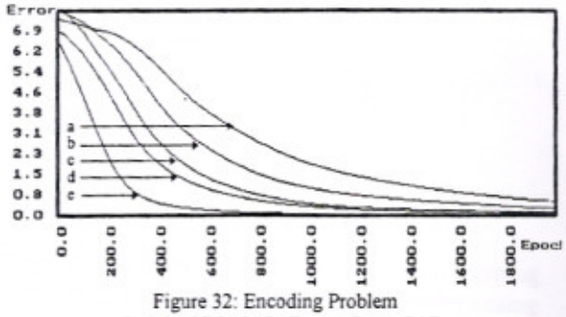


Figure 32: Encoding Problem
a:Standard BP b: SAB c: Super SAB
d:J(5, 3)-CF (η) e: J(5, 3)-J(5, 3)-CDF(η, λ)

Remark 6: Adaptive steepness (ASBP) method is a method [11] which uses gradient descent rule for adaptation of steepness parameter. In this method each neuron k has steepness parameter λ_k, which is changed by following rule:

$$\Delta \lambda_k = -\varepsilon \frac{\partial E}{\partial \lambda_k}$$

Figure 34 compares the performance of ASBP and class **D** schemes for parity problem. To the authors knowledge, ASBP method is the only method for adaptation of steepness parameter, reported in the literature.

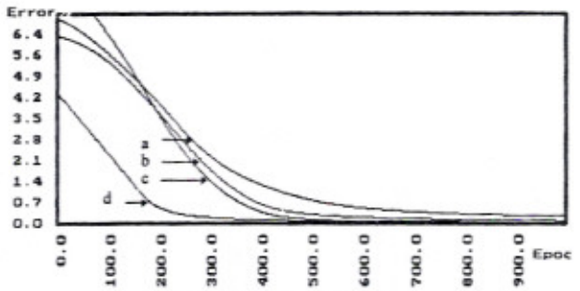


Figure 33: Digit problem

a:Standard BP b: SAB c: Super SAB d:J(5, 3)-CF (η)

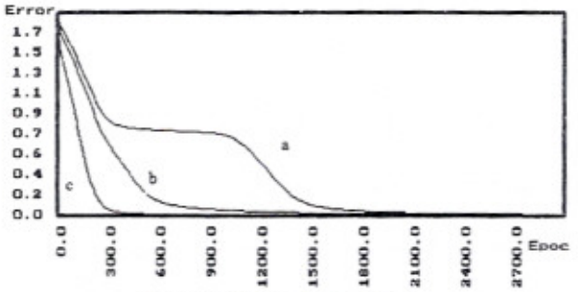


Figure 34: Parity problem

a: Standard BP b: ASBP c: J(2, 1)-DF(λ)

6. Conclusion

Dynamic learning rate adjustment could be indirectly or directly. Momentum and conjugate gradient methods are example of indirect methods. Since indirect methods haven't been satisfactory enough in many cases direct adjustment of the learning rate have been proposed. Direct methods can be classified into two groups: local methods, such as Bold Driver method [9] in which one learning rate is used for all the weights and global methods such as SAB [20] and Super SAB [23]methods in which one learning rate is used for each weight.

In this paper two new classes of direct methods called ***B*** and ***C*** classes and a new class of indirect method called ***D*** class are presented. Since class ***B*** schemes adapt two different learning rates one for hidden layer and one for output layer it may be called quasi-global method. All the schemes in these classes uses learning automata of fixed or variable structure type to adapt BP parameter. The adaptation is based on the error surface behavior.

With the introduction of these new classes of schemes we propose a new classification tree for learning rate adjustment methods. Figure 28 shows this new classification.

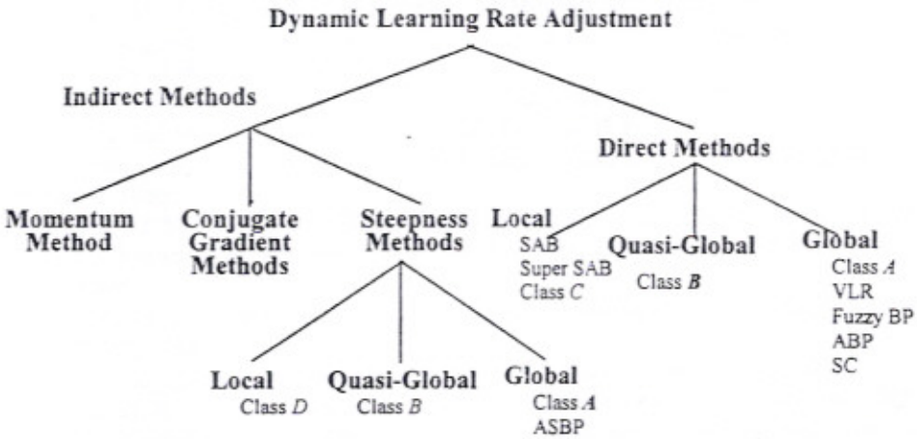


Figure 28 : Classification tree for dynamic learning rate adjustment schemes

To evaluate the performance of these methods, simulation studies were carried out on several learning problems with different error surfaces. Simulations indicate that dynamic adaptation of BP parameters using the proposed methods increase the speed of convergence of the standard BP and superior to most previous schemes reported for adaptation of BP parameters.

7. References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation" , In Parallel distributed processing, Cambridge, MA: MIT Press, 1986.

[2] P. Arabshahi, J. J. Choi, R. J. Marks, and T. P. Caudell, "Fuzzy Control of Back propagation", Proc. of IEEE Int. Conf. on Fuzzy Systems, pp. 967-972, 1992.

[3] N. Kandil, K. Khorasani, R. V. Patel, and V. K. Sood, "Optimum Learning Rate for Back propagation Neural Networks", In Neural Networks Theory, Technology, and Applications, Edited by P. K. Simpson, pp. 249-251, 1996.

[4] A. G. Parlos, B. Fernadez, A. F. Atya, J. Muthusami, and W. K. Tsai, "An Accelerated Learning Algorithm for Multi-Layer Preceptron Networks", IEEE Trans. on Neural Networks, Vol. 5, No. 3, pp. 493-497, 1994.

[5] J. P. Cater, "Successfully Using Peak Learning Rates of 10 (and Greater) in Backpropagation Networks with the Heuristic Learning Algorithm", IEEE Proc. of First Int. Conf. on Neural Networks, Vol. II, pp. 645-651, 1987.

[6] M. A. Franzini, "Speech Recognition with Backpropagation", IEEE Proc. of Ninth Annual Conf. on Eng. in Medicine and Biology, pp. 1702-1703, 1987.

[7] T. P. Vosl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, "Accelerating the Convergence of Backpropagation Method", Biological Cybernetics, pp. 257-263, 1987.

[8] G. Tesauro and B. Janssens, "Scaling Relationships in Backpropagation Learning", Complex Systems. pp. 39-44, 1988.

[9] D. Sarkar, "Methods to Speedup Error Backpropagation Learning Algorithm", ACM Computing Surveys, No. 4, Vol. 27, Dec. 1995.

[10] M. Riedmiller and B. Heinrich, "A Direct Method for Faster Backpropagation Algorithm", Neural Networks, Vol. 5, pp. 465-471, 1992.

[11] A. Sperduti and A. Starita, "Speed Up Learning and Network Optimization with Extended Backpropagation", Neural Networks, Vol. 6, pp. 365-383, 1993.

[12] M. B. Menhaj and M. R. Meybodi, "A Novel Learning Scheme for Feedforward Neural Nets", Proc. of ICEE-95, University of Science and Technology, Tehran, Iran, 1994.

[13] M. B. Menhaj and M. R. Meybodi, "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, Dec. 1996.

[14] M. B. Menhaj and M. R. Meybodi, "Application of Learning Automata to Neural Networks ", Proc. of Second Annual CSI Computer Conference CSIC'96, Tehran, Iran , pp. 209-220, Dec. 1996.

[15] M. B. Menhaj and M. R. Meybodi, "Using Learning Automata in Backpropagation Algorithm with Momentum", Technical Report, Computer Eng. Department, Amirkabir University of Technology, Tehran, Iran , 1997.

[16] H. Beigy and M. R. Meybodi, " Adaptation of Momentum Factor and Steepness parameter in Backpropagation Algorithm Using Fixed Structure Learning Automata", Proc. of 4th Int. Annual Conf. of Computer Society of Iran, pp. 117-124, 1999.

[17] H. Beigy , M. R. Meybodi, and M. B. Menhaj, "Adaptation of Learning Rate in Backpropagation Algorithm Using Fixed Structure Learning Automata", Proc. of ICEE-98.

[18] K. S. Narendra and M. A. L. Thathachar, Learning Automata: An Introduction, Prentice-Hall, Englewood cliffs, 1989.

[19] M. B. Menhaj and M. H. Hagen, "Rapid Learning Using Modified Backpropagation Algorithms for Multi-Layer Feedforward Neural Nets", Proc. of ICEE-95, University of Science and Technology, Tehran, Iran, 1995.

[20] R. A. Jacobs, "Increased Rates of Convergence Through Learning Rate Adaptation", Neural Networks, Vol. 1, pp. 295-307, 1988.

[21] L. E. Scales, Introduction to Non-linear Optimization, New York:Springer-verlag, 1985.

[22] M. R. Devos and G. A. Orban, "Self Learning Backpropagation", Proc. of NeuroNimes, 1988.

[23] T. Tollenaere, "SuperSAB: Fast Adaptive Backpropagation with Good Scaling Properties", Neural Networks, Vol. 3, 1990.

[24] H. Hsin, C. C. Li, M. Sun, and R. J. Sclabani, "An Adaptive Training algorithm for BP Neural Network", IEEE Trans. on System, Man and Cybern. Vol. 25, No. 3, pp. 512-514, 1995.

[25] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combination of Genetic Algorithms and Neural Networks: A Survey of the State of the Art", In Proc. COGANN-92, Int. Workshops on Combination of Genetic Algorithms and Neural Networks, 1992.

[26] M. R. Meybodi and S. Lakshmivarhan, "Optimality of a General Class of Learning Algorithm", Information Science, Vol. 28, pp. 1-20, 1982.

[27] M. R. Meybodi and S. Lakshmivarhan, "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.

[28] M. R. Meybodi, "Results on Strongly Absoulutely Expedient Learning Automata", Proc. of OU Inference Conf. 86, ed. D. R. Mootes and R. Butrick, Athens, Ohio: Ohio University Press, pp. 197-204, 1987.

ITRC

The 7th Iranian Conference On Electrical Engineering

Iran Telecommunication Research Center 17-19 May 1999

Proceedings

Computer