International Journal of Neural Systems, Vol. 12, No. 1 (2002) 45–67
© World Scientific Publishing Company

NEW LEARNING AUTOMATA BASED ALGORITHMS FOR ADAPTATION OF BACKPROPAGATION ALGORITHM PARAMETERS

M. R. MEYBODI[*] and H. BEIGY[†]
Soft Computing Laboratory, Computer Engineering Department,
Amirkabir University of Technology, Tehran, Iran
[*] *meybodi@ce.aku.ac.ir*
[†] *beigy@ce.aku.ac.ir*

Received 28 December 2000
Revised 13 August 2001
Accepted 13 August 2001

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 effects 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 the proposed schemes is its ability to escape from local minima with high possibility during the training period. The feasibility of proposed methods is shown through simulations on several problems.

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 past experiences. For example, if the learning rate is too small, convergence can be very slow; if too large, paralysis and continuous 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 backpropagation 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-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] have 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 also been 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 that not only increases the rate of convergence of the network but it increases the likelihood of bypassing the local minimum. 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, one learning automaton is assigned to every link or every neuron in the network for determining the parameters for that link or neuron is there schemes. As the characteristics of the error surface may be unique in every dimension, different learning rate or steepness parameter may be required for every link and/or every neuron. Using learning automata as the adaptation technique, the search for optimum is carried out in probability space rather than in parameter space as in the case with other adaptation algorithms. In the standard gradient method, the new operation point lies within a neighborhood distance of the previous point. This is not the case for adaptation algorithm based on stochastic principles, as the new operating point is determined by probability function and is therefore not considered to be near the previous operating point. This gives the algorithm a higher ability to locate the global optimum. The simulation results show the feasibility of the proposed methods and their superiority to the existing LA based schemes. The proposed schemes have two important aspects: higher speed of convergence and higher probability of escaping from the local minima. In order to evaluate the performance of proposed schemes, simulations are carried out on eight learning problems: digit recognition, odd parity, encoding, symmetry, classification of sonar signals, vowel recognition, printed Farsi digit recognition, and printed Farsi character recognition problems and the results are compared with results obtained from standard BP and some of the learning methods for adaptation of BP parameters. These problems are chosen because they possess different error surfaces and collectively present an environment that is suitable to determine the effect of the proposed methods.

8×8 *Dot Numeric Font Learning*: There are numbers 0.9, and each represented by an 8×8 grid of black and white dot as shown in Fig. 1. The network must learn to distinguish these numbers. The training set consists of 10 patterns. 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.

***Three-Bit Odd-Parity Problem*:** In this problem, a string of three inputs is applied to the network, the output of network is zero (one) if the number of ones in the input is odd (even).[1] The training set consists of 8 patterns. This network has three input units, three hidden units, and one output unit.

***Encoding Problem*:** In this problem, a set of orthogonal input patterns is mapped to a set of orthogonal output patterns through a small set of hidden units.[1] The training set consists of 8 patterns. The network architecture used for solving this problem consists of 8 input units, 3 hidden units, and 8 output units.

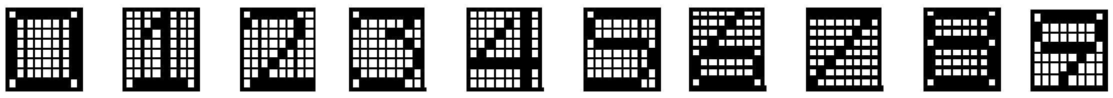


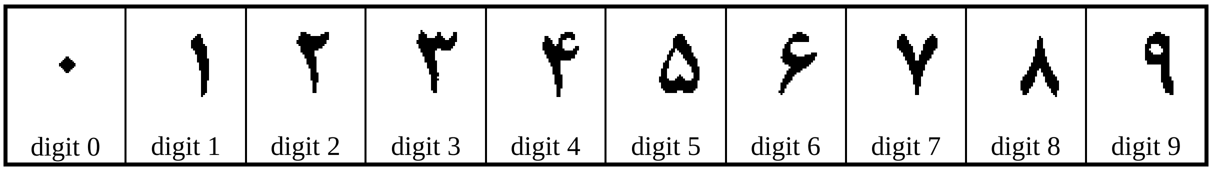
Fig. 1.

***Symmetry Problem*:** This problem classifies input string as to whether or not they are symmetric about center.[1] The training set consists of 64 patterns. The network architecture used consists of 8 input units, 2 hidden units, and 1 output unit.

Classification of Sonar Signals: The task is to train a neural network to discriminate between sonar signal bounced off a metal cylinder and those bounced off a roughly cylindrical rock.[29] The training set consists of 104-member 60-dimensional patterns. The network that must learn to distinguish mine from rock has 60 input units, 24 hidden units, and 2 outputs, one indicating a cylinder and the other a rock.

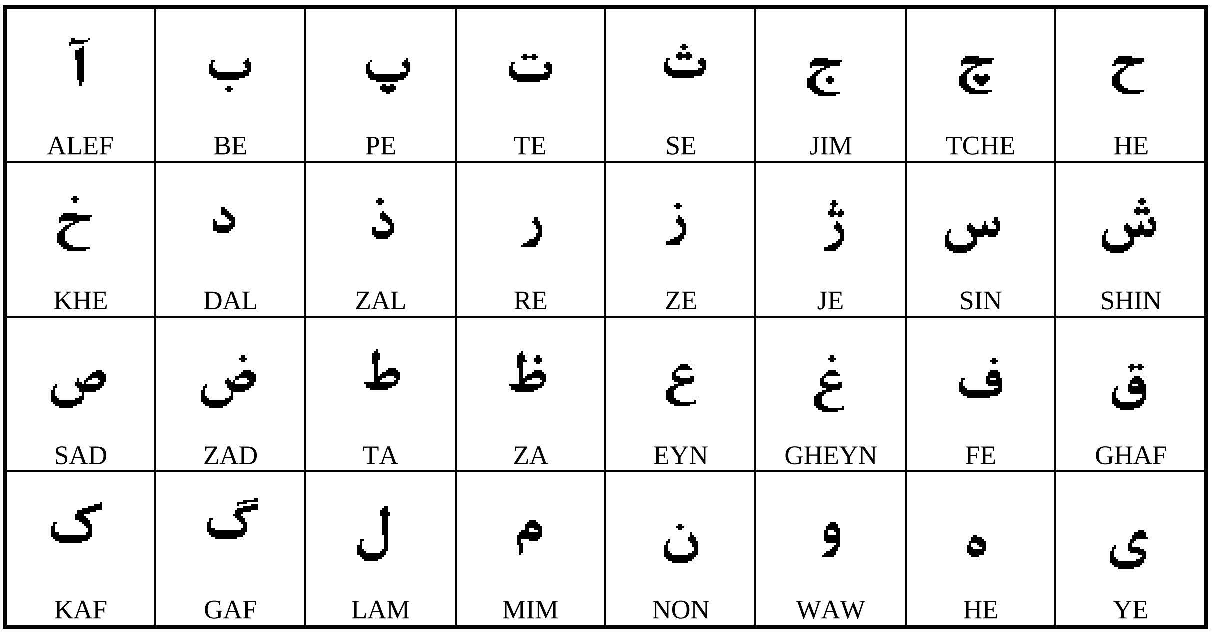
Vowel Recognition: In this problem, we have eleven steady state vowels of British English. These vowels are from 15 different speakers. Each vowel is represented by a set of LPC derived log area ratios. The training set consists of 110 patterns. The network architecture used for recognition of vowels have 10 input units, 22 hidden units, and 11 output units.[33]

Printed Farsi Digit Recognition: The ten printed Farsi digits are shown in Fig. 2. There is a page of 170 printed Farsi digits, 17 samples for every digit.[32] 160 samples are used to train the network and the remaining samples are used for testing purpose. This page is digitized with a resolution of 300dpi. The momentum constants M_1 through M_7 are extracted from digitized images and submitted as inputs to the neural network. The network that must learn to classify these digits has 7 input nodes, 30 hidden units, and 10 output units, one for each digit.



۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
digit 0	digit 1	digit 2	digit 3	digit 4	digit 5	digit 6	digit 7	digit 8	digit 9

Fig. 2.



آ	ب	پ	ت	ث	ج	چ	ح
ALEF	BE	PE	TE	SE	JIM	TCHE	HE
خ	د	ذ	ر	ز	ژ	س	ش
KHE	DAL	ZAL	RE	ZE	JE	SIN	SHIN
ص	ض	ط	ظ	ع	غ	ف	ق
SAD	ZAD	TA	ZA	EYN	GHEYN	FE	GHAF
ک	گ	ل	م	ن	و	ه	ی
KAF	GAF	LAM	MIM	NON	WAW	HE	YE

Fig. 3.

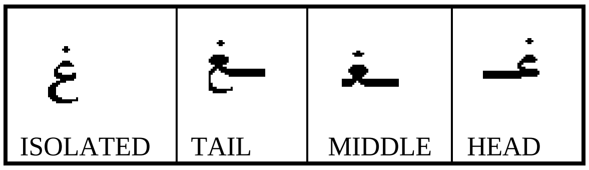


Fig. 4.

Printed Farsi Character Recognition: The Farsi alphabet consists of 32 basic characters shown in Fig. 3. These characters differ from other systems of characters in their structure and in the way they connect to form words. The same character may take different shapes according to its position in the word. For example, the character "GHEYN" has four different shapes according to its appearance at the head, the middle, the tail, or the isolated. This feature increases the number of Farsi characters to more than 90 different shapes, including all forms of 32 basic characters, numeric characters, and punctuation characters.[32] A page of 32 isolated Farsi characters is scanned with resolution of 300dpi. Each character has 5 samples. The momentum constants M_1 through M_7 are used as inputs to the neural network. The network architecture used for this problem consists of 7 inputs which are connected to 31 hidden units and 32 output units.

The rest of the paper is organized as follows: Sec. 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 Sec. 3. Section 4 presents the proposed learning automata based schemes. The simulation results are given in Sec. 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:

$$W(n+1)=W(n)+\eta G(n)+\alpha[W(n)-W(n-1)], \tag{1}$$

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

$$G(n) = -\nabla E_p(n)\,. \tag{2}$$

E_p is the sum of squared error given by:

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

where $T_{p,j}$ and $O_{p,j}$ are desired and actual outputs for pattern p at output node j. A major problem 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 of required iterations at a reasonable number. One of the 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 continuous and differentiable. The activation function is normally a sigmoid function chosen between $1/1+\exp(-\lambda\text{net})$ and tanh (λnet). The coefficient of the exponent of the exponential term determines the steepness 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 is not very close to zero. This makes the BP algorithm to 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 changes 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 a great assistance in speeding up convergence along shallow gradients, allowing the path the network takes towards the solution to pick up 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 nearer the minima and to be large further 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 a quintuple ($\underline{\alpha}$, $\underline{\phi}$, $\underline{\beta}$, F, G) where:

- $\alpha = (\alpha_1, \ldots, \alpha_R)$ is the set of actions that it must choose from.
- $\Phi = (\Phi_1, \ldots, \Phi_s)$ is the set of states.
- $\beta = \{0, 1\}$ is the set of inputs where 1 represents a penalty and 0 represents a reward.
- F: $\Phi \times \beta \to \Phi$ is a map called the transition map. It defines the transition of the state of the automaton on receiving input, F may be stochastic.
- G: $\Phi \to \alpha$ is the output map and determines the action taken by the automaton if it is in state Φ_j.

The selected action serves as an 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 penalizes (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 time $(n + 1)$. Note that $\{c_i\}$ is 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 probabilities of the transition from one state to another and probabilities of correspondence of action and state are fixed, the automaton is said to be fixed-structure automata and otherwise the automaton is said to be variable-structure automata. We summarize some of the 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 automaton that uses this strategy is refered as $L_{2,2}$ where the first subscript refers to the number of states and the second subscript is the number of actions.

The two-action automata with memory ($L_{2N,2}$): 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. While the automaton $L_{2,2}$ switches from one action to another on receiving a failure response from the environment, $L_{2N,2}$ 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; the automaton switches from one action to the other. 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 automaton is said to have a total memory of $2N$. For every favorable response, the state of automaton moves deeper into the memory of corresponding action, and for an unfavorable response, moves out of it. This automaton can be extended to multiple action automata. The state transition graph of $L_{2N,2}$ automaton is shown in Fig. 5.

***The Krinsky automata*:** This automaton behaves exactly like $L_{2N,2}$ automaton when the response of turehe environment is unfavorable, but for favorable response, any state ϕ_i (for $i = 1, \ldots, N$) 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

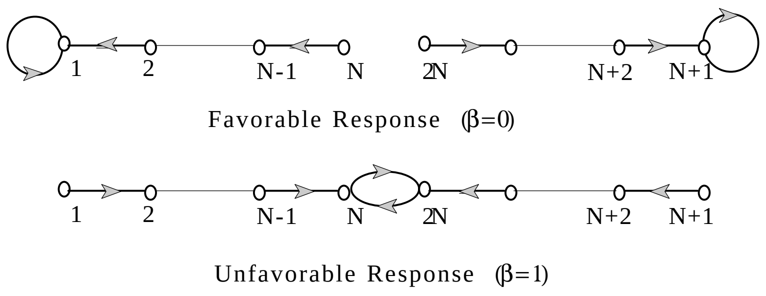


Fig. 5. The state transition graph for $L_{2N,2}$.

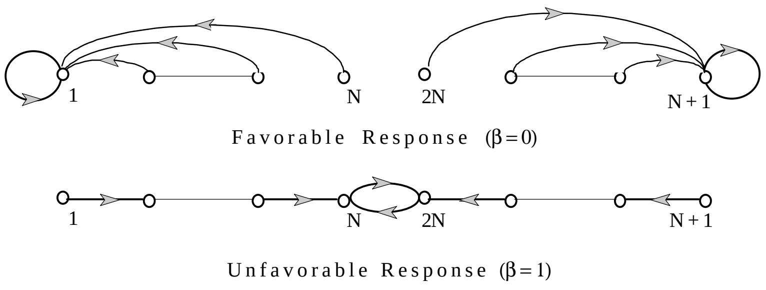


Fig. 6. The state transition graph for Krinsky Automaton.

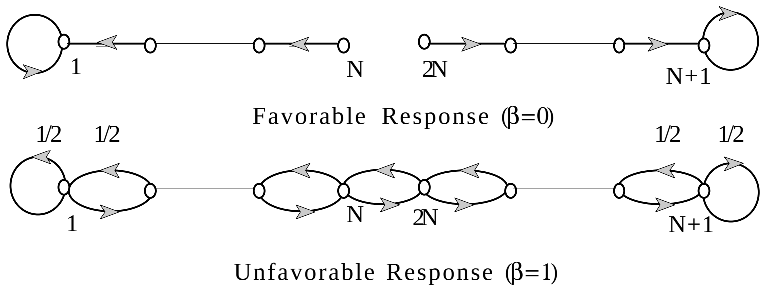


Fig. 7. The state transition graph for Krylov Automaton.

of N consecutive unfavorable responses are needed to change from one action to another. The state transition graph of Krinsky automaton is shown in Fig. 6.

***The Krylov automaton*:** This automaton has state transition that is identical to the $L_{2N,2}$ automaton when the output of the environment is favorable. However, when the response of the environment is unfavorable, a state ϕ_i ($i \neq 1$, 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 $i = N$, automaton state moves ϕ_{N-1} to ϕ_{2N} and with the same probability 0.5. When $i = 2N$, automaton state moves ϕ_{2N-1} to ϕ_N with the same probability 0.5. The state transition graph of Krylov automaton is shown in Fig. 7.

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

***Variable-structure Automata*:** Variable-structure automaton is represented by sextuple $\langle \underline{\beta}, \underline{\phi}, \underline{\alpha}, \underline{P}, G, T\rangle$, where $\underline{\beta}$ a set of inputs actions, ϕ 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 the 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

$$p_j(k) = \begin{cases} p_j(k) + a(1 - p_j(k)) & \text{if } i = j \\ p_j(k)(1-a) & \text{if } i \neq j \end{cases} \tag{4}$$

if β is zero and P is unchanged if β is one. The parameter a, which is called step length, 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

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

if $\beta(k) = 0$ and

$$p_j(k) = \begin{cases} p_j(k)(1-b) & \text{if } i = j \\ \dfrac{b}{r-1} + (1-b)p_j(k) & \text{if } i \neq j \end{cases} \tag{6}$$

if $\beta(k) = 1$. The parameters a and b represent reward and penalty parameters, respectively. The parameter $a(b)$ determines the amount of increase (decreases) of the action probabilities.

Learning automata have been used in many applications such as: solving NP problems,[34,35] optimization of structure of neural networks,[36,37] queuing theory,[38] game theory,[39,40] telephone traffic control,[44] pattern recognition,[45] control of ATM networks,[41–43] to mention a few. For more information about learning automata, refer to Refs. 18, 26–28, 41, 46 and 47.

3. LA Based Schemes for Adaptation of BP Parameters

In this section, first we 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 automata have been associated to the network. The learning automata (or automata) based on the observation of the random response of the neural network, adapted one or more of BP parameters. The interconnection of learning automata and neural network is shown in Fig. 8. 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 adjust the parameters of BP algorithm. The actions of the automata correspond to the values of the parameters being calculated and input to the automata is some functions of the error in the output of neural network.

A function of error between the desired and the actual outputs of network is considered as the response of the environment. A window on the past values of the errors is swiped and the average

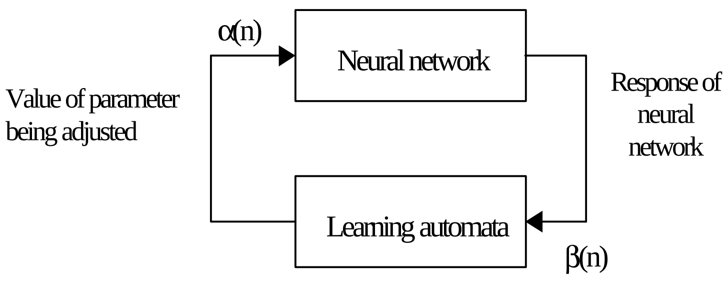


Fig. 8. The interconnection of learning automata and neural network.

value of the error in this window computed. If the difference of the average value in the last two steps is less than a predefined threshold value, the response of the environment is favorable and if this difference is greater than the threshold value, the response of the environment is unfavorable.

The existing LA based procedures for adaptation of BP parameters can be classified into two classes, which we call class ***A*** and class ***B***. In class ***A*** schemes, one automaton is used for the whole network[12,14] whereas in class ***B*** schemes, separate automata, such is one for each layer (hidden and output), are used.[13,15–17] Classes ***A*** and ***B*** depending on the type of automaton being used (fixed or variable structure) each can be classified into two subclasses. The parameter adapted by class ***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 class ***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***. Without loss of generality, we assume that in class ***A*** and class ***B***, the neural network has one hidden layer.

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

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

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

The letters **F** and **V** in the above names denote 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 parameters increases the rate of convergence by a large amount. Figure 9 borrowed from Ref. 17 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 automaton, the reward and penalty coefficients 0.001 and 0.0001 are chosen. It has been 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

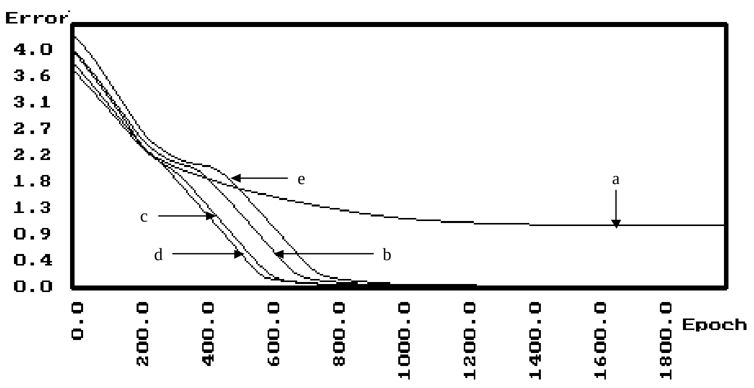


Fig. 9. Performance of different class A based schemes. (a) Standard BP (b) Tsetline(4, 4)-AF(η) (c) Krinsky(2, 4)-AF(η) (d) Krylov(2, 4)-AF(η) (e) LR-P-AV(η).

a fixed predetermined learning rate or momentum factor.[15,16]

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 class ***D***. In a class ***C*** scheme, one automaton is associated to each link of the network in order to adjust the parameter for that link. In class ***D*** scheme, one automata is associated to each neuron of the network in order to adjust the parameter for that neuron. Class ***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 the 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 class ***D*** schemes, the automaton receives favorable response from the environment if the algebraic sign of derivative in the two consecutive iterations is the same and receives unfavorable response if the algebraic sign of the derivative in the two consecutive iterations alternates. The algorithms given in Figs. 10 and 11 describe class ***C*** and class ***D*** schemes, respectively.

```
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 l^th layer do
          for all weights w for n^th node in l^th layer do
            if $\mathrm{Sign}\left(\frac{\partial E_p(k)}{\partial w}\right) = \mathrm{Sign}\left(\frac{\partial E_p(k-1)}{\partial w}\right)$ then
              //The sign at iteration k and k-1 is the same
                $\eta$ = call Automata (0)      // $\beta$ is 0
            else
                $\eta$ = call Automata (1)      // $\beta$ 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 number
end procedure
```

Fig. 10. 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 $\beta$ in l^th layer do
          if $\mathrm{Sign}\left(\frac{\partial E_p(k)}{\partial \lambda}\right) = \mathrm{Sign}\left(\frac{\partial E_p(k-1)}{\partial \lambda}\right)$ then
          //The sign at iteration k and k-1 is the same
             $\lambda$ =call Automata (0)          // $\beta$ is 0
          else
             $\lambda$ =call Automata (1)          // $\beta$ is 1
          end if
        end for
      end for
    end for
    eall UpdateWeights
     // Batch weights updating is used
  until k > N   // N is maximum training epoch number
end procedure
```

Fig. 11. BP with a class ***D*** scheme.

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

$$\mathrm{net}_I^L = \sum_{k=0}^{N_{L-I}} W_{K,I}^L \times O_K^{L-1} \tag{7}$$

O_K^L is the output of kth neuron in Lth layer given by $O_K^L = f_K^L(\mathrm{net}_K^L)$, where f_K^L shows the activation function of kth neuron in Lth layer that is one of sigmoidal or tanh functions. Differentiation of E with respect λ_K^L yields:

$$\frac{\partial E_p(n)}{\partial \lambda_K^L} = -\delta_K^L \times \frac{\partial f}{\partial \lambda_K^L} \tag{8}$$

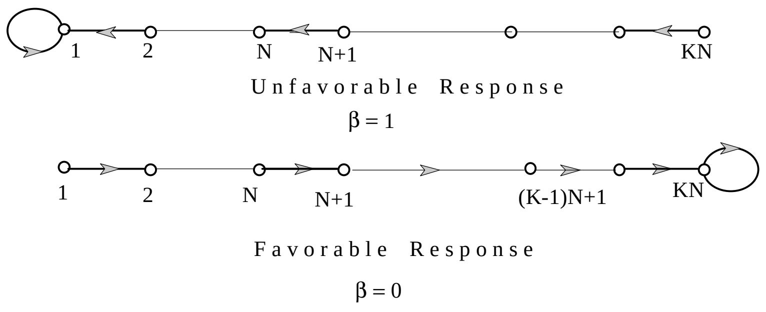


Fig. 12. The state transition graph for $J_{KN,K}$.

where

$$\delta_K^L = \begin{cases} (T_K - O_K) & \text{if } L \text{ is output layer} \\ \sum_{J=1}^{N_{L+1}} \delta_J^{L+1} \times W_{K,J}^{L+1} \times \dfrac{\partial f}{\partial \text{net}_J^{L+1}} & \text{if } L \text{ is hidden layer} \end{cases} . \tag{9}$$

A deterministic fixed structure learning automaton: In what is to follow, we introduce a new fixed structure learning automaton called *J*-automata. This automaton can be used by class ***C*** and class ***D*** schemes to achieve a higher degree of performance comparing to the other known schemes in class ***C*** or class ***D***. The proposed automaton which we denote it by $J_{KN,K}$ has *KN* states and K actions. This automaton 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. State transition graph of this automaton for favorable response and unfavorable response is shown in Fig. 12.

The environment produces the favorable response if the algebraic sign of the derivative in two consecutive iterations is the same and unfavorable response is produced by the environment if the algebraic sign of the derivative in two consecutive iterations alternates. When this automaton is used to adjust the learning rate (steepness parameter), each action corresponds to one of the values of the learning rate (steepness parameter). The automaton, on reward moves to the higher number state (hoping to eventually increase the value of the parameter) and on penalty moves to the lower number state (hoping to eventually decrease 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 iterations will the automaton change its action (switches from one value for the related parameter to another value). Therefore, N is the memory of the depth associated with each action and automaton is said to have a total memory of *KN*. If parameter μ can assume K values, then $\mu_1 \leq \mu_2 \leq \cdots \leq \mu_K$ states $(k-1)N+1$ through *kN* which corresponds 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 to increase if in N consecutive iterations, the sign of the derivative does not change and decreases if in N consecutive iterations, the sign of derivative alternates.

Remark 1

Variable learning rate (VLR) scheme: In this remark, first we explain the Variable learning rate (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 continues to go down, then learning rate will increase and convergence will speed up.

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

$$\nabla E = \left[\frac{\partial E}{\partial \pi_1}, \frac{\partial E}{\partial \pi_2}, \ldots, \frac{\partial E}{\partial \pi_n},\right]^T . \tag{10}$$

Gradient vector ∇E extends in the direction of the greatest rate of change of E but it does not mean that all $\nabla_k E$ (the kth element of ∇E) have the same algebraic sign. In other words the greatest rate of increase(decrease) of E does not 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 used by BP to walk along all directions, which may create oscillation along some axis and leads to a lower rate of convergence. On the contrary in $\mathbf{C}(\boldsymbol{D})$ 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 a 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 algorithm given in Fig. 13 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 Eq. (1). The changes in the weight at iteration of n is given by:

$$\Delta W(n) = \eta G(n) + \alpha \Delta W(n-1) \tag{11}$$

```
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$\left(\frac{\partial E_p(k)}{\partial w}\right)$ = Sign$\left(\frac{\partial E_p(k-1)}{\partial w}\right)$ then
              //The sign at iteration k and k-1 is the same
                $\eta$ = call Automata1 (0)        // $\beta$ is 0
            else
                $\eta$ = call Automata1 (1)        // $\beta$ is 1
            end if
          end for
        end for
        for all steepness parameters $\lambda$ in l^th layer do
          if Sign$\left(\frac{\partial E_p(k)}{\partial \lambda}\right)$ = Sign$\left(\frac{\partial E_p(k-1)}{\partial \lambda}\right)$ then
      //The sign at iteration k and k-1 is the same
            $\lambda$ = call Automata2 (0)       // $\beta$ is 0
          else
            $\lambda$ = call Automata2 (1)       // $\beta$ 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 number
end procedure
```

Fig. 13. Automata1-Automata2-CDF(η, λ) scheme.

Solving the difference equation (11) gives the following time series equation

$$\Delta W(n) = \eta \sum_{t=0}^{n} \alpha^{n-1} G(t) . \tag{12}$$

By inspection of Eq. (12), 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 $0 \leq |\alpha| < 1$.
2. When G has the same algebraic sign on consecutive iterations, $|\Delta W|$ grows, and W is

```
procedure Simultaneous_C_Scheme_BP (Automata1, Automata2)
  Initialize the weights to small random values.
  Initialize the parameters for automata 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 $\mathrm{Sign}\left(\frac{\partial E_p(k)}{\partial w}\right) = \mathrm{Sign}\left(\frac{\partial E_p(k-1)}{\partial w}\right)$ then
              //The sign at iteration k and k-1 is the same
                $\eta$ = call Automata1 (0)      // $\beta$ is 0
                $\alpha$ = call Automata2 (0)      // $\beta$ is 0
            else
                $\eta$ = call Automata1 (1)      // $\beta$ is 1
                $\alpha$ =call Automata2 (1)      // $\beta$ 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 number
end procedure
```

Fig. 14. Automata1-Automata2-CF(η, α) scheme.

adjusted by a large amount. Hence, the inclusion of momentum term accelerates the convergence of algorithm. To accelerate more, the momentum factor must have a value as large as possible.

3. When the algebraic sign of G alternates in consecutive iterations, $|\Delta W|$ shrinks and W is adjusted by a small amount. Hence, the inclusion of momentum term stabilizes the convergence of algorithm. To accelerate more, the momentum factor must have a value as small 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 algorithm given in Fig. 14 describes the simultaneous adaptation of learning rate and momentum factor. This scheme is denoted by **Automata1-Automata2-CF**(η, α).

5. Simulations

Typical simulations for four previously mentioned problems for different parameter adaptation schemes are shown in Figs. 15–20. Figure 15 compares the

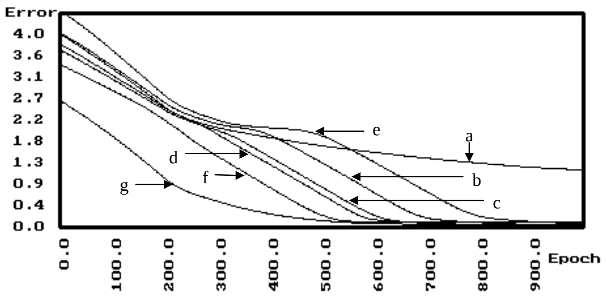


Fig. 15. Digit problem. (a) Standard BP (b) Tsetline(4,4)-AF(η) (c) Krinsky$(2,4)$-AF(η) (d) Krylov$(2,4)$-AF(η) (e) L_{R-P}-AV(η) (f) VLR (g) J$(2,1)$-CF(η).

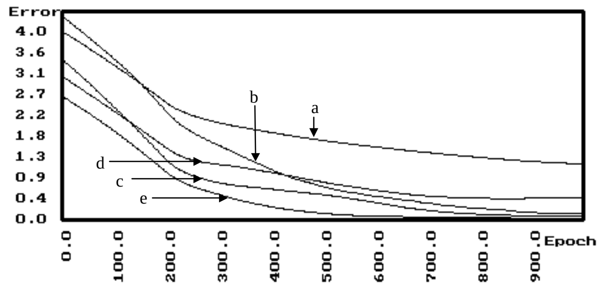


Fig. 16. Digit problem. (a) Standard BP (b) Tsetline(4, 4)-CF(η) (c) Krinsky(2, 4)-CF(η) (d) Krylov(2, 4)-CF(η) (e) J(2, 1)-CF(η).

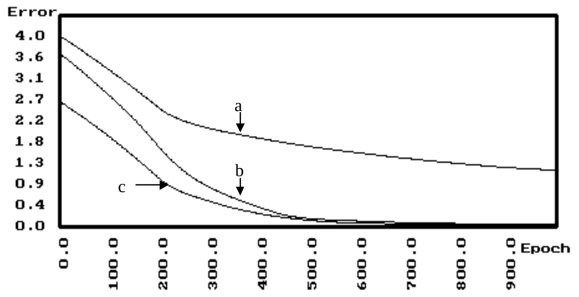


Fig. 17. Digit problem. (a) Standard BP (b) Tsetline(4, 6)-Tsetline(2, 4)-BF(η) (c) J(10, 1)-CF(η).

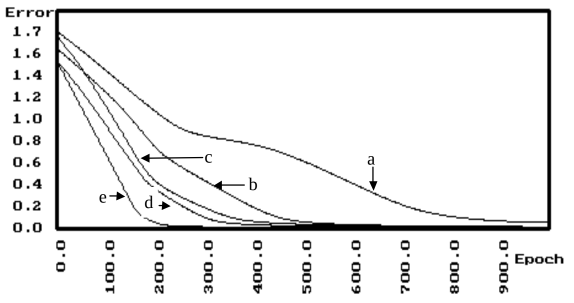


Fig. 18. Parity problem. (a) Standard BP (b) VLR (c) J(2, 1)-CF(η) (d) J(2, 1)-CF(η) (e) J(2, 1)-J(2, 1)-CDF(η, λ).

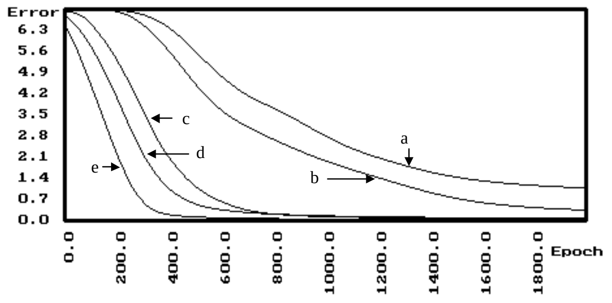


Fig. 19. Encoding problem. (a) Standard BP (b) VLR (c) J(2, 1)-CF(η) (d) J(5, 6)-CF(λ) (e) J(2, 1)-J(5, 6)-CDF(η, λ).

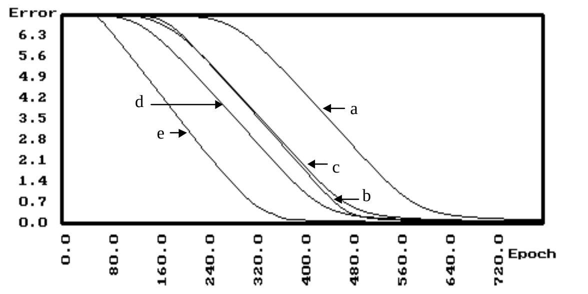


Fig. 20. Symmetry problem. (a) Standard BP (b) VLR (c) J(5, 6)-CF(λ) (d) J(2, 1)-CF (η) (e) J(2, 1)-J(5, 6)-CDF(η, λ).

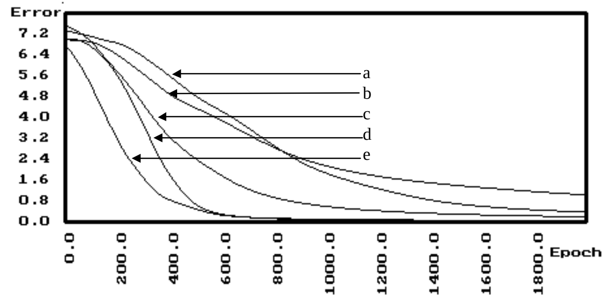


Fig. 21. Encoding problem. (a) SC Scheme (b) Standard BP (c) VLR Scheme (d) Fuzzy BP (e) J(2, 1)-J(5, 6)-CDF(μ, λ).

performance of different class ***A*** schemes with J-CF scheme. Figure 16 indicates that J-CF scheme has a higher speed of convergence than any known scheme in class ***C***. Figure 17 compares the J-CF scheme with the best scheme in class ***B***. Figures 18–20 indicate that if μ and λ are adapted simultaneously, the performance of the BP algorithm increases by a large amount. Figures 21–23 compare the performance of J-J-CDF (μ, λ) scheme with VLR scheme and schemes proposed by Arabshahi (FuzzyBP) and Darken and Moody (SC). For all simulations, we have taken the momentum factor (α) to be zero and the parameters of different schemes are chosen in such a way that the best performance will

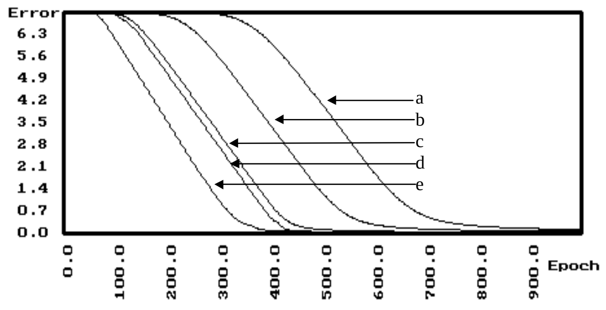


Fig. 22. Symmetry problem. (a) SC Scheme (b) Standard BP (c) VLR Scheme (d) Fuzzy BP (e) J(2, 1)-J(5, 6)-CDF(μ, λ).

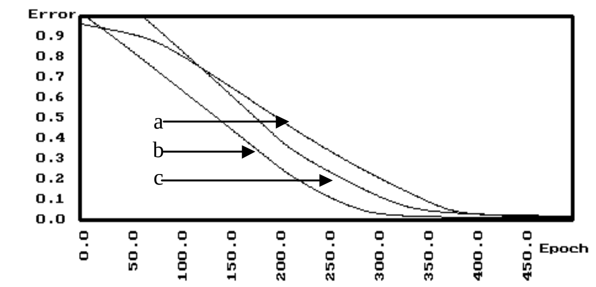


Fig. 25. Parity problem. (a) J(5, 10)-J(5, 10)-CDF(η, λ) (b) J(5, 10)-J(5, 10)-CF(η, α) (c) J(5, 10)-CF(η) with fixed momentum factor.

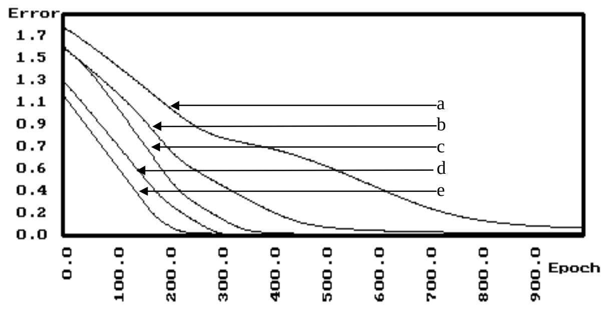


Fig. 23. Parity problem. (a) SC Scheme (b) Standard BP (c) VLR Scheme (d) Fuzzy BP (e) J(2, 1)-J(2, 1)-CDF(μ, λ).

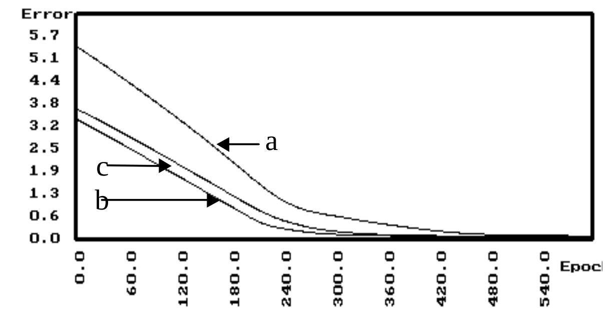


Fig. 26. Encoding problem. (a) J(5, 10)-J(5, 6)-CDF(η, λ) (b) J(5, 10)-J(5, 10)-CF(η, α) (c) J(5, 10)-CF(η) with fixed momentum factor.

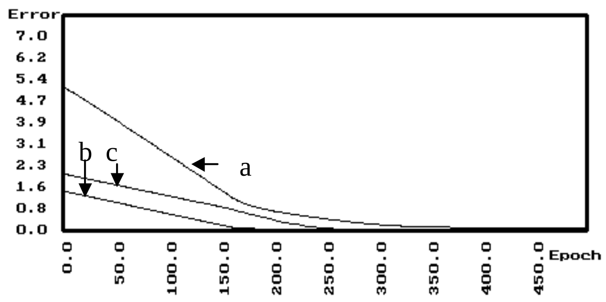


Fig. 24. Digit problem. (a) J(5, 10)-CF(η) (b) J(5, 10)-J(5, 10)-CF(η, α) (c) J(5, 10)-CF(η) with constant momentum factor.

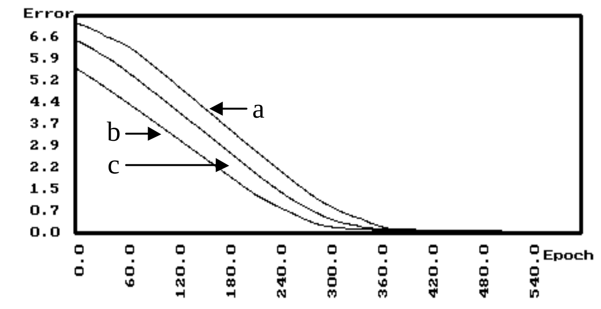


Fig. 27. Symmetry problem. (a) J(15, 10)-J(5, 4)CDF(η, λ) (b) J(15, 10)-J(15, 10)-CF(η, α) (c) J(15, 10)-CF(η) with fixed momentum factor.

be obtained. The plot for each simulation is averaged to be over 200 runs.

Figures 24–27 show the performance of different schemes when both learning rate and momentum factor are adapted. The plot for each simulation is averaged to be over 200 runs.

Remark 2

J-CF(η) scheme has a close relationship with Jacobs 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 the 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 automaton for adaptation of learning parameter, and with the definition of favorable and unfavorable response given before, and also inspecting the transition graphs for these automata, 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 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 28 shows the effect of memory depth on the speed of learning.

Remark 4

J-CF(η) and **J-DF**(λ) schemes become 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 are very small and for a certain amount of changes in the value of the parameter, the automata needs to make large number of states change. This effect is similar to the effect we observed for large memory depth and small number of actions. Figure 29 shows the effect of the number of actions on the speed of learning.

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

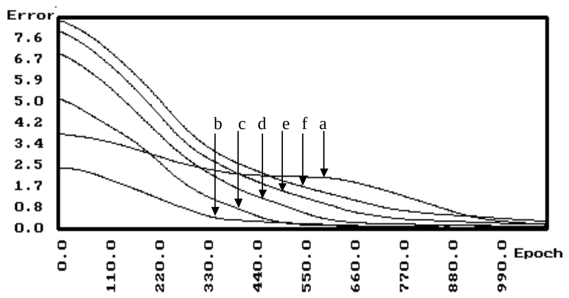


Fig. 29. Digit problem. (a) Standard BP (b) J(10, 10)-CF(η) (c) J(250, 10)-CF(η) (d) J(500, 10)-CF(η) (e) J(750, 10)-CF(η) (f) J(1000, 10)-CF(η).

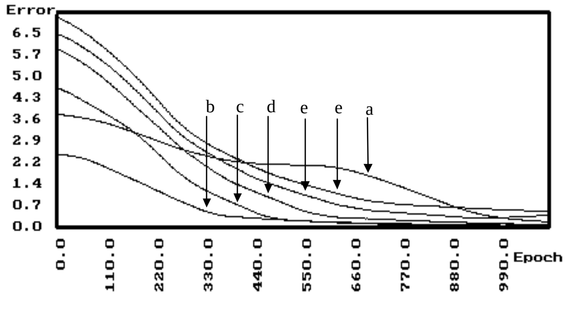


Fig. 28. Digit problem. (a) Standard BP (b) J(10, 10)-CF(η) (c) J(10, 250)-CF(η) (d) J(10, 500)-CF(η) (e) J(10, 750)-CF(η) (f) J(10, 1000)-CF(η).

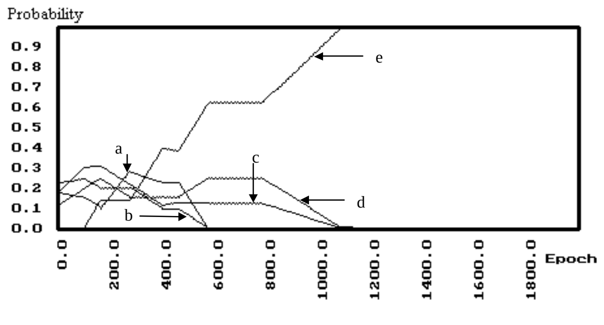


Fig. 30. Action probability of J(5, 2)_DF(λ) scheme for parity problem. (a) Action 1 (b) Action 2 (c) Action 3 (d) Action 4 (e) Action 5.

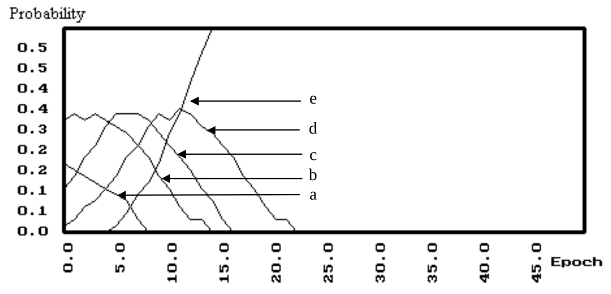


Fig. 31. Action probability of J(5, 2)_CF(η) scheme for parity problem. (a) Action 1 (b) Action 2 (c) Action 3 (d) Action 4 (e) Action 5.

randomly selected neuron (weight) is plotted. As shown in Figs 30 and 31, the system converges to the action with the higher value. This effect has also been observed in genetic algorithms for the determination of BP parameters.[25] The values of actions are 0.4, 0.8, 1.2, 1.6, and 2.

Remark 6

Self-Adaptive BP (SAB) was developed independently by Jacobs[20] and Devious 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 is increased if in two consecutive iterations, the gradient has the same sign and is decreased if the 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 the sign of gradient alternates, the learning rate

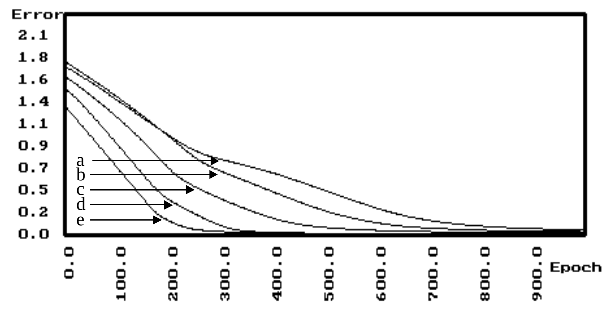


Fig. 32. Parity problem. (a) Standard BP (b) SAB (c) SuperSAB (d) J(5, 3)-CF (η) (e) J(5, 3)-J(5, 3)-CDF(η, λ).

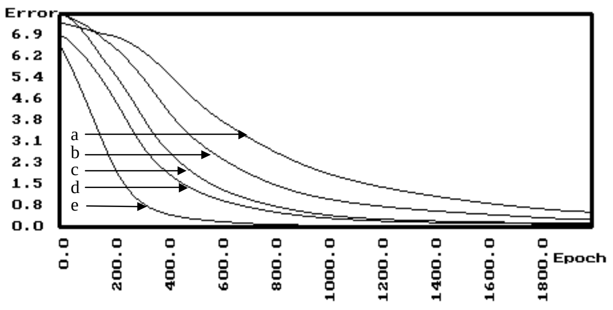


Fig. 33. Encoding problem. (a) Standard BP (b) SAB (c) Super SAB (d) J(5, 3)-CF(η) (e) J(5, 3)-J(5, 3)-CDF(η, λ).

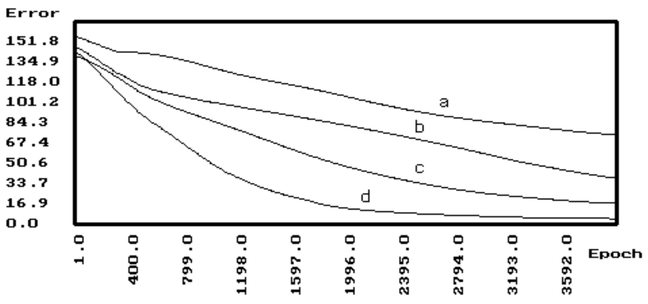


Fig. 34. Farsi digit recognition. (a) Standard BP (b) SAB (c) Super SAB (d) J(10, 10)-CF(η).

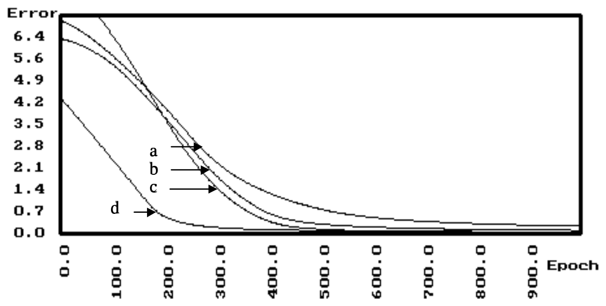


Fig. 35. Digit problem. (a) Standard BP (b) SAB (c) Super SAB (d) J(5, 3)-CF(η).

is reset to initial value. SuperSAB algorithm that is proposed by Tollenaere[23] overcomes these problems.

These two schemes are simulated on given problems and compared with the standard BP, SAB, SuperSAB, J-CF, and J- CDF schemes. The simulation results that are given in Figs. 32–40 show the superiority of J-CF and J-CDF schemes. The plot for each simulation is averaged to be over 200 runs.

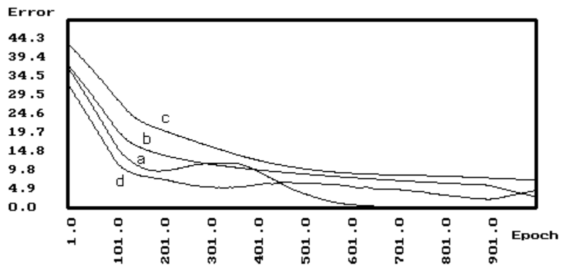


Fig. 36. Sonar signal recognition. (a) Standard BP (b) SAB (c) Super SAB (d) J(4, 4)-CF(η).

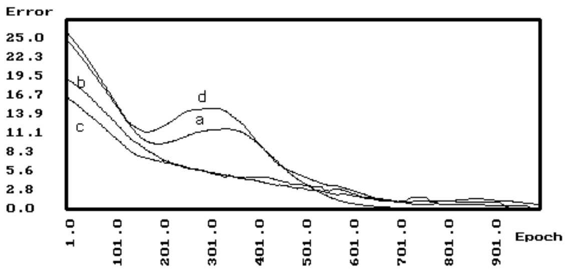


Fig. 37. Sonar signal recognition. (a) J(4, 4)-CF(η) (b) Krinsky(4, 4)-CF(η) (c) Tsetline (4, 4)-CF(η) (d) Krylov(4, 4)-CF(η).

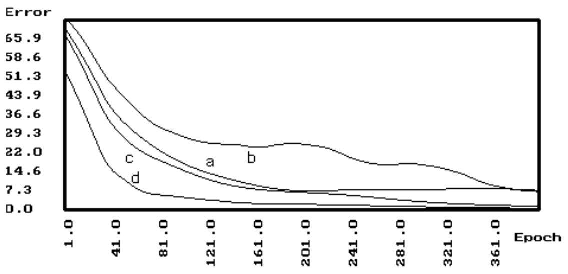


Fig. 38. Vowel recognition. (a) Standard BP (b) SAB (c) Super SAB (d) J(2, 8)-CF(η).

Remark 7

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

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

Figures 41–45 compare the performance of ASBP and class **D** schemes for parity problem. To the

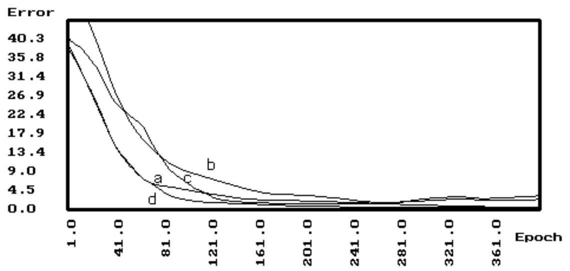


Fig. 39. Vowel recognition. (a) J(2, 8)-CF(η) (b) Krinsky(2, 8)-CF(η) (c) Tsetline (2, 8)-CF(η) (d) Krylov(2, 8)-CF(η).

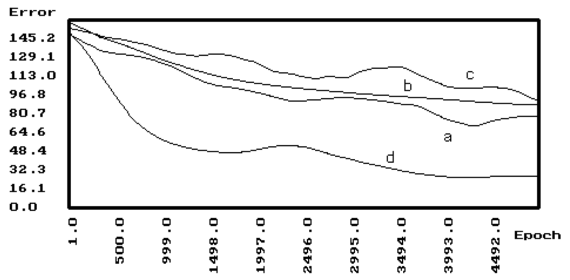


Fig. 40. Vowel recognition. (a) Standard BP (b) SAB (c) Super SAB (d) J(10, 20)-CF(η).

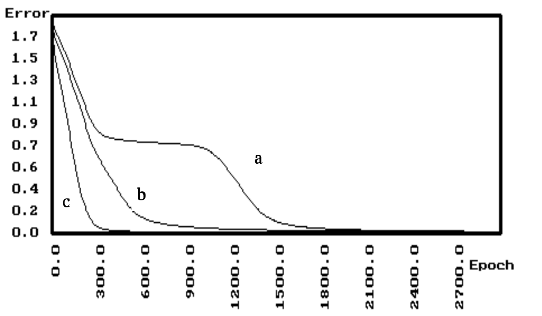


Fig. 41. Parity problem. (a) Standard BP (b) ASBP (c) J(2, 1)-DF(λ).

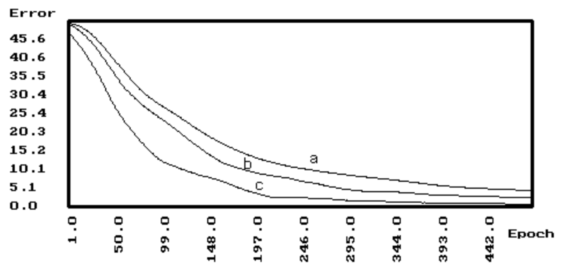


Fig. 42. Sonar signal recognition. (a) ASBP (b) J(5, 6)-DF(λ) (c) J(10, 20)-J(5, 6)-CDF(η, λ).

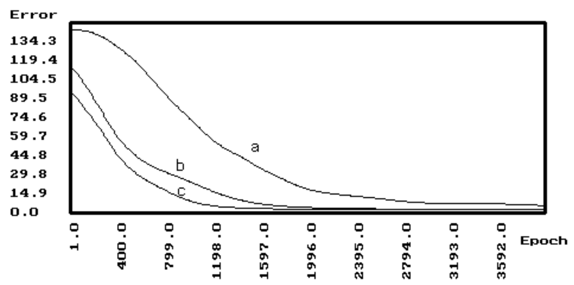


Fig. 43. Farsi digit recognition. (a) ASBP (b) J(5, 6)-DF(λ) (c) J(10, 20)-J(5, 6)-CDF(η, λ).

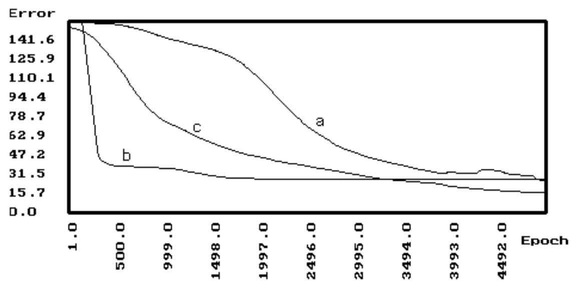


Fig. 44. Farsi character recognition. (a) ASBP (b) J(5, 6)-DF(λ) (c) J(10, 20)-J(5, 6)-CDF(η, λ).

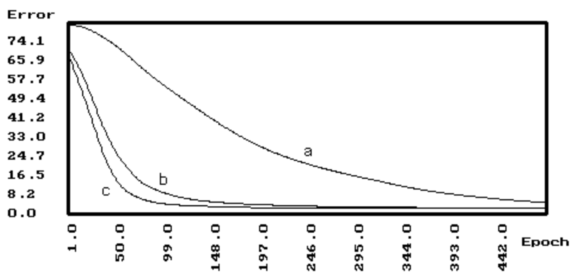


Fig. 45. Vowel recognition. (a) ASBP (b) J(5, 6)-DF(λ) (c) J(10, 20)-J(5, 6)-CDF(η, λ).

authors' knowledge, ASBP method is the only method for adaptation of steepness parameter, reported in the literature. The plot for each simulation is averaged to be over 200 runs.

6. LA Based Schemes and Local Minima

In this section, we examine the ability of the learning automata based schemes to escape from local minima. For this purpose, we chose a problem in which local minima occurred frequently.[30] This example considers the sigmoidal network for the XOR Boolean function with the quadratic cost function and the standard learning environment. The training set of this problem is given in Table 1.

The network which is used has two input nodes x and y, two hidden units, and one output unit. In this problem, if the hidden units produce the

Table 1.

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

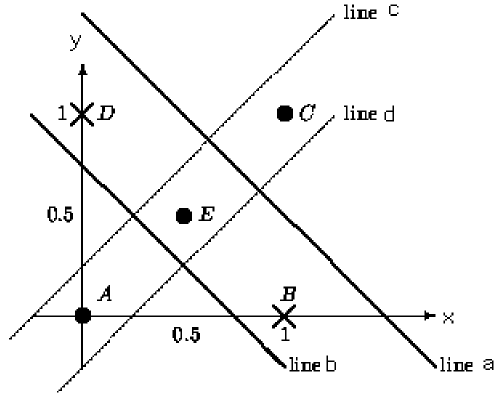


Fig. 46. Lines produced by hidden units of neural network.

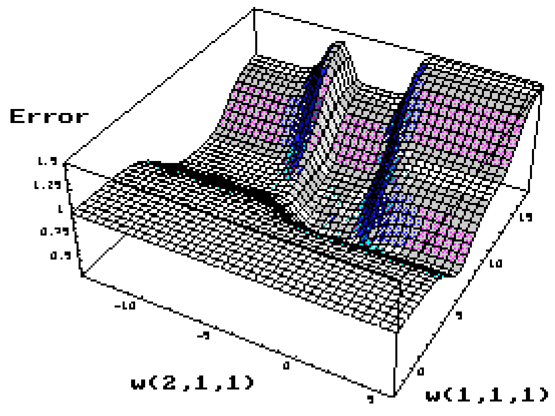


Fig. 47. Error surface as a function of weights $w_{2,1,1}$ and $w_{1,1,1}$.

Table 2.

Algorithm	Class	Not Converged	Converged
BP		20	0
SAB		20	0
SuperSAB		20	0
VLR		18	2
FuzzyBP		18	2
ASBP		13	7
Tsetline-AF(μ)	A	18	2
Krinsky-AF(μ)	A	14	6
Krylov-AF(μ)	A	17	3
LR-P-AF(μ)	A	16	4
Tsetline-AF(λ)	A	7	13
Tsetline-TsetlineG-BF(μ)	B	18	2
Tsetline-Krylov-BF(μ)	B	18	2
Tsetline-Krinsky-BF(μ)	B	15	5
Tsetline-Tsetline-BF(μ)	B	15	5
J-DF(λ)	D	15	5
J-CF(μ)	C	13	7
J-J-CDF(μ, λ)	CD	8	12

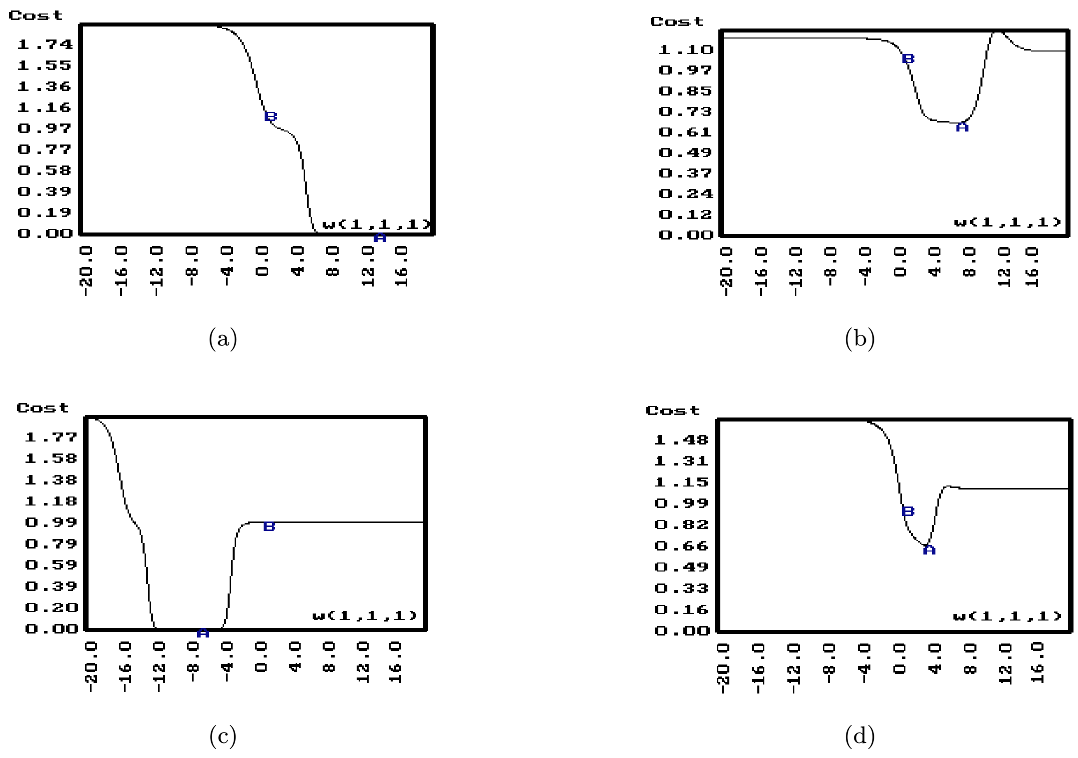


Fig. 48. (a) J-J-CDF(μ, λ) (converged to global minima) (b) J-DF(λ) (stuck at local minima) (c) J-CF(μ) (converged to global minima) (d) BP algorithm (stuck at local minima).

lines a and b, the local minima has occurred and if hidden units produce the lines c and d, the global minima occurred.[31] Figure 46 shows these configurations. The error surface of the network as a function of weights $w_{2,1,1}$ and $w_{1,1,1}$ is given in Fig. 47.

Depending on the initial weights, the gradient can get stuck in points where the error is far from being zero. The presence of these local minima is intuitively related to the symmetry of the learning environment. Experimental evidence of the presence of local minima is given in Fig. 47.

To show the superiority of LA based adaptation algorithm in terms of escaping from local minima, we test twelve different LA based algorithms, 4 from class ***A***, 4 from class ***B***, 1 from class ***C***, 1 from class ***D***, and 1 from class ***CD***, and compare their results with standard BP and five other known adaptation methods: SAB,[20] SuperSAB,[23] VLR method,[19] ASBP method,[11] and fuzzy BP.[2] In all simulations, each learning automaton has K actions equally spaced in interval (0,1]. Window size, threshold value, and maximum number of epochs are chosen to be 10, 0.01, and 32000, respectively. The result of simulation for 20 runs is summarized in Table 2. Note that for standard BP and also for standard BP when SAB or SuperSAB method is used to adapt the learning rate, none of the 20 runs converges to the global minima. Among the non-LA based methods, the ASBP method performs the best. For this scheme, 7 out of 20 runs converges to global minima which is comparable to some of the LA based schemes we have tested. The best result is obtained for algorithm Tsetline-AF(μ) for which 13 out of 20 runs converges to global minimum. The next best result belongs to J-J-CDF(μ, λ) scheme. Figures 48(a)–(d) show some typical runs. Each run uses a random initial point near the local minima. In these figures, the initial point is denoted by letter "B" and the converged point is denoted by "A". The curves in these figures are obtained by projecting the error surface on axis $w_{1,1,1}$.

The reason for such a good performance of LA based schemes is that in the standard gradient method, the new operation point lies within a neighborhood distance of the previous point. This is not the case for adaptation algorithm based on stochastic principles, as the new operating point is determined by probability function and is therefore not considered to be near the previous operating point. This gives the algorithm a higher ability to locate the global optimum. In general, the LA approach for optimization has two distinct advantages over the classical hill climbing methods: (1) the parameter space need not be metric and (2) since the search space is conducted in the path probability space than parameter space, a global rather than a local optimum can be found.

7. Time and Space Complexity of LA-Based Adaptation Schemes

In this section, we discuss the time and storage overhead imposed on BP algorithm when LA based adaptation schemes are used. For the implementation of FSLA, three memory locations are needed in order to keep track of the state, number of actions, and memory depth of the automata. Changing state and also realizing the action associated with the state of the automata at each epoch requires few comparison, integer subtraction and addition. Therefore, the storage and time overhead imposed by each FSLA is $\theta(1)$. This leads to θ(M) storage and θ(M) time overhead for ***CF*** type schemes and θ(N) storage and θ(N) time overhead for ***DF*** type schemes, where M and N are the number of weights and number of neurons in the network, respectively. For these schemes, in addition to storage needed to implement FSLA, storage for previous sign of gradient, adapted parameter, and reinforcement signal β for each neuron are needed. When VSLA with K actions are used, the storage needed by ***CV*** and ***DV*** type schemes are θ(KM) and θ(KN), respectively. This is because of the storage required by each automaton to store its action probability vector P. Due to updating the action probability vector P by the automata at each epoch, the time

Table 3. The time and space overhead of proposed schemes.

Algorithm	Storage Overhead	Time Overhead
AV	θ(K)	θ(K)
AF	θ(1)	θ(1)
BV	θ(K)	θ(K)
BF	θ(1)	θ(1)
CV	θ(KM)	θ(KM)
CF	θ(M)	θ(M)
DV	θ(KN)	θ(KN)
DF	θ(N)	θ(N)
SAB	θ(M)	θ(M)
SuperSAB	θ(M)	θ(M)

Table 4. Ratio of execution time of different schemes to standard BP.

Algorithm	Ratio of Execution Time
SAB	1.66
SuperSAB	1.58
VLR	1.79
CLASS C	1.56
CLASS D	1.55
CLASS CD	1.56

overhead for $\boldsymbol{CV}$ and $\boldsymbol{DV}$ type schemes are θ(KM) and θ(KN), respectively. Class ***A*** and ***B*** schemes have an overhead of $\theta(1)$ for both time and storage if FSLA is used and overhead of $\theta(K)$ for both time and space if VSLA is used. Table 3 summarizes the storage and time overhead imposed by different schemes.

In order to justify the overheads imposed by the proposed schemes, the ratio of execution time of SAB, SuperSAB, QPROP, RPROP, VLR, and different LA based schemes to the execution time of standard BP algorithm for parity problem are measured and given in Table 4.

8. Conclusion

Learning rate can be adjusted directly or indirectly. Momentum, conjugate gradient, and class **D** are examples of indirect methods. Since the indirect methods haven't been satisfactory enough in many cases, the direct adjustment of learning rate have been proposed. Direct methods can be classified in three classes: global method, local method, and quasi-global method. In global methods such as Bold Driver and Class **A** schemes, one learning rate is used for all weights. In quasi-global methods such as class **B** schemes, one learning rate is used for a set of weights, and in local methods such as SAB, SuperSAB, and class **C** schemes, one learning rate is used for each weight in the network.

With the introduction of these new classes of schemes, we propose a new classification tree for learning rate adjustment methods. Figure 49 shows this new classification.

In this paper, a new class of direct methods called class ***C*** and a new class of indirect method called class ***D*** are presented. All the schemes in these classes use learning automata of fixed or variable structure type to adapt BP parameter. The adaptation is based on the error surface behavior. 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 are superior to most previous schemes reported for adaptation of BP parameters.

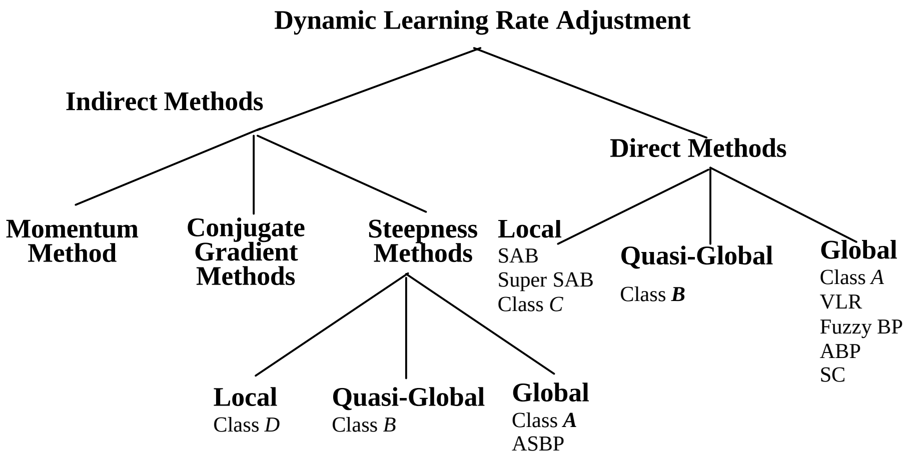


Fig. 49. Classification tree for dynamic learning rate adjustment schemes.

References

1. D. E. Rumelhart, G. E. Hinton and R. J. Williams 1986, "Learning internal representations by error propagation," in *Parallel Distributed Processing* (MIT Press, Cambridge, MA).
2. P. Arabshahi 1996, "Fuzzy parameter adaptation in optimization: Some neural net training examples," *IEEE Computational Science and Engineering*, 57–65.
3. N. Kandil, K. Khorasani, R. V. Patel and V. K. Sood 1996, "Optimum learning rate for back propagation neural networks," in *Neural Networks Theory, Technology, and Applications*, eds. P. K. Simpson, pp. 249–251.
4. A. G. Parlos, B. Fernadez, A. F. Atya, J. Muthusami and W. K. Tsai 1994, "An accelerated learning algorithm for multi-layer preceptron networks," *IEEE Trans. on Neural Networks* **5**, 493–497.
5. J. P. Cater 1987, "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.
6. M. A. Franzini 1987, "Speech recognition with backpropagation," *IEEE Proc. of Ninth Annual Conf. on Engineering in Medicine and Biology*, pp. 1702–1703.
7. T. P. Vosl, J. K. Mangis, A. K. Rigler, W. T. Zink and D. L. Alkon 1987, "Accelerating the convergence of backpropagation method," *Biological Cybernetics*, 257–263.
8. G. Tesauro and B. Janssens 1988, "Scaling relationships in backpropagation learning," *Complex Systems*, 39–44.
9. D. Sarkar 1995, "Methods to speedup error backpropagation learning algorithm," *ACM Computing Surveys* **27**, 519–542.
10. M. Riedmiller and B. Heinrich 1992, "A direct method for faster backpropagation algorithm," *Neural Networks* **5**, 465–471.
11. A. Sperduti and A. Starita 1995, "Speedup learning and network optimization with extended backpropagation," *Neural Networks* **6**, 365–383.
12. M. B. Menhaj and M. R. Meybodi 1995, "A novel learning scheme for feedforward neural nets," *Proc. of 3rd Iranian Conf. on Electrical Engineering, ICEE-95*, University of Science and Technology, Tehran, Iran.
13. M. B. Menhaj and M. R. Meybodi 1996, "Flexible sigmodal type functions for neural nets using game of automata," *Proc. of Second Annual Computer Society of Iran Computer Conference CSICC-96*, Amirkabir University of Technology, Tehran, Iran, 221–232.
14. M. B. Menhaj and M. R. Meybodi 1996, "Application of learning automata to neural networks," *Proc. of Second Annual Computer Society of Iran Computer Conference CSICC-96*, Amirkabir University of Technology, Tehran, Iran, 209–220.
15. M. B. Menhaj and M. R. Meybodi 1997, "Using learning automata in backpropagation algorithm with momentum," *Technical Report*, Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran.
16. H. Beigy and M. R. Meybodi 1999, "Adaptation of momentum factor and steepness parameter in backpropagation algorithm using fixed structure learning automata," *Proc. of 4th Annual Computer Society of Iran Computer Conference CSICC-99*, Sharif University of Technology, Tehran, Iran, 117–124.
17. H. Beigy, M. R. Meybodi and M. B. Menhaj 1998, "Adaptation of learning rate in backpropagation algorithm using fixed structure learning automata," *Proc. of 6th Iranian Conf. on Electrical Engineering, ICEE-95*, K. N. Tosi University of Technology, Tehran, Iran, 117–123.
18. K. S. Narendra and M. A. L. Thathachar 1989, *Learning Automata: An Introduction* (Prentice-Hall, Englewood cliffs).
19. M. B. Menhaj and M. H. Hagen 1995, "Rapid learning using modified backpropagation algorithms for multi-layer feedforward neural nets," *Proc. of 3rd Iranian Conf. on Electrical Engineering, ICEE-95*, University of Science and Technology, Tehran, Iran.
20. R. A. Jacobs 1988, "Increased rates of convergence through learning rate adaptation," *Neural Networks* **1**, 295–307.
21. L. E. Scales 1985, *Introduction to Non-linear Optimization* (Springer-Verlag, New York).
22. M. R. Devos and G. A. Orban 1988, "Self learning backpropagation," *Proc. of NeuroNimes.*
23. T. Tollenaere 1990, "SuperSAB: Fast adaptive backpropagation with good scaling properties," *Neural Networks* **3**.
24. H. Hsin, C. C. Li, M. Sun and R. J. Sclabani 1995, "An adaptive training algorithm for BP neural network," *IEEE Trans. on System, Man and Cybern.* **25**, 512–514.
25. J. D. Schaffer, D. Whitley and L. J. Eshelman 1992, "Combination of genetic algorithms and neural networks: A survey of the state of the art," *Proc. of Int. Workshops on Combination of Genetic Algorithms and Neural Networks, COGANN-92*, 1–37.
26. M. R. Meybodi and S. Lakshmivarahan 1982, "Optimality of a general class of learning algorithms," *Information Science* **28**, 1–20.
27. M. R. Meybodi and S. Lakshmivarhan 1984, "On a class of learning algorithms which have a symmetric behavior under success and failure," *Springer-Verlag Lecture Notes in Statistics*, pp. 145–155.
28. M. R. Meybodi 1987, "Results on strongly absoulutely expedient learning automata," *Proc. of OU Inference Conf. 86*, eds. D. R. Mootes and R. Butrick (Athens, Ohio: Ohio University Press), pp. 197–204.
29. R. P. Gorman and T. J. Sejnowski 1988, "Analysis of

hidden units in a layered network trained to classify sonar targets," *Neural Networks* **1**, 75–89.

30. M. Gori and A. Tesi 1992, "On the problem of local minima in backpropagation," *IEEE Trans. on Pattern Analysis and Machine Intelligence* **14**, 76–86.
31. P. Frasconi, M. Gori and A. Tesi 1992, "Success and failures of backpropagation: A theoretical investigation," *Technical Report*, Dipartimento di Sistemi e Information, Universita di Firenze, Firenze, Italy.
32. V. Dastpak 1992, "Automatic recognition of Farsi printed letters," *Ms. Thesis*, Computer Eng. Dept. Amirkabir University of Technology, Tehran, Iran (in Persian).
33. D. H. Deterding 1989, "Speaker normalisation for automatic speech recognition," *Ph.D. Thesis*, University of Cambridge, UK.
34. B. J. Oommen and E. V. de St. Croix 1996, "Graph partitioning using learning automata," *IEEE Trans. on Computers* **45**, 195–208.
35. H. Beigy and M. R. Meybodi 1999, "Graph isomorphism using learning automata," *Technical Report 1CE99*, Computer Eng. Dept. Amirkabir University of Technology, Tehran, Iran (in Persian).
36. H. Beigy and M. R. Meybodi 1999, "Optimization of topology of neural networks using learning automata," *Proc. of 4th Annual Int. Computer Society of Iran Computer Conf. CSICC-98*, Tehran, Iran, 417–428 (in Persian).
37. M. R. Meybodi and H. Beigy 2001, "Neural network engineering using learning automata: Determination of desired size for three layer feedforward neural network," *Journal of Faculty of Engineering* **34**(4), March 2001, University of Tehran press, 1–26.
38. M. R. Meybodi and S. Lakshmivarhan 1983, "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, 106–109.
39. S. Lakshmivarahan and K. S. Narendra 1981, "Learning algorithms for two-person zero-sum stochastic games with incomplete information," *Mathematics of Operations Research* **6**, 379–386.
40. S. Lakshmivarahan and K. S. Narendra 1982, "Learning algorithms for two-person zero-sum stochastic games with incomplete information: A unified approach," *SIAM J. Control and Optimization* **20**, 541–552.
41. P. Mars, J. R. Chen and R. Nambiar 1998, *Learning Algorithms: Theory and Application in Signal Processing, Control, and Communications* (CRC press New York).
42. A. V. Vasilakos and S. A. Koubias 1988, "On routing and performance comparison of techniques for packet switched networks using learning automata," *IEEE Int. Sympos. on Circuits and Systems*, pp. 109–113.
43. A. V. Vasilakos and A. F. Atlasis 1994, "LB-SELA: Rate based access control for ATM networks," *Proc. of INFOCOM-94*, pp. 1552–1559.
44. P. R. Srikantakumar and K. S. Narendra 1982, "A learning model for routing in telephone networks," *SIAM J. of Control and Optimization* **20**, 34–57.
45. A. G. Barto and P. Anandan 1985, "Pattern-Recognizing stochastic learning automata," *IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-15*, pp. 360–375.
46. S. Lakshmivarahan 1981, *Learning Algorithms: Theory and Applications* (Springer-Verlag, New York).
47. K. Najim and A. S. Pozyak 1994, *Learning Automata: Theory and Applications* (Pergamon Press, Oxford).