PAPER

Introducing an Adaptive VLR Algorithm Using Learning Automata for Multilayer Perceptron

Behbood MASHOUFI[†], **Mohammad Bagher MENHAJ**[†*], **Sayed A. MOTAMEDI**[†], *and* **Mohammad R. MEYBODI**[††], *Nonmembers*

SUMMARY One of the biggest limitations of BP algorithm is its low rate of convergence. The Variable Learning Rate (VLR) algorithm represents one of the well-known techniques that enhance the performance of the BP. Because the VLR parameters have important influence on its performance, we use learning automata (LA) to adjust them. The proposed algorithm named Adaptive Variable Learning Rate (AVLR) algorithm dynamically tunes the VLR parameters by learning automata according to the error changes. Simulation results on some practical problems such as sinusoidal function approximation, nonlinear system identification, phoneme recognition, Persian printed letter recognition helped us better to judge the merit of the proposed AVLR method.

key words: *multilayer neural network, backpropagation, variable learning rate, learning automata*

1. Introduction

Consider the backpropagation (BP) learning algorithm with the batch method. For the purpose of obtaining an optimal weight vector in an iterative manner, a descent type algorithm was first developed by Werbos in 1974, and rediscovered by Parker in 1982 and once again by Rumelhart in 1986 [1]. Unfortunately, it has been observed that convergence rate of the BP is extremely slow, especially for the networks with more than one hidden layer. The intrinsic reason behind this is the saturation property of the activation function used for the neurons. Once the output of a unit lies in the saturation area, the corresponding descent gradient would take a very small value if the output error is very large. This will result in a very little progress in the weight adjustment if one takes a fixed small learning rate parameter. To avoid this undesired phenomenon, one may consider relatively large learning rate. This would be dangerous, however, because it may lead to divergence of the iteration especially when the weight adjustment happens to fall into the surface regions with a large steepness.

Therefore, an efficient learning algorithm is sought to dynamically tune its learning rate in accordance with variations of the gradient values [10]. Research into dynamic changes of the learning rate of the BP algorithm has been reported in [5], [6]. Basically, they all dynamically increase or decrease the learning rate by a fixed factor as well as the momentum based on the observation of error signals. Some other acceleration methods have also been presented, including modifications of optimization criterion and use of second order methods (e.g., the Newton method [7], the Broyden-Fletcher-Goldfarb-Shanno and the Levenberg-Marquardt methods et al. [8], [9]). For improving neural networks' performance, some researchers have combined learning automata (LA) and neural networks [11]–[18]. M.L. Tsetlin and his co-workers started work on learning automaton in the 1960s in the Soviet Union [19]. The Variable structure learning automata (VSLA) or the Fixed Structure Learning Automata (FSLA) have been recently used to find the appropriate values of the BP training algorithm's parameters. Baba, Handa and Sato used a hierarchical structure stochastic automata (HSSA) to find the appropriate values of parameters for the BP training algorithm [15], [16].

This paper considered different methods that are dynamically tuned the learning rate. The paper furthermore described and compared the Variable Learning Rate (VLR) algorithm and the Learning Automaton (LA) based learning rate adaptation algorithms with each other. Because the VLR parameters have an important influence on its performance, we use learning automata to adjust them. In the proposed algorithm named Adaptive Variable Learning Rate (AVLR) algorithm, the VLR parameters are tuned dynamically according to error variations by learning automata. Simulation results on sinusoidal function approximation, odd parity, symmetry, digit recognition, nonlinear system identification, phoneme recognition and Persian printed letter recognition problems are very promising so that the merit of the proposed AVLR can be easily judged.

The rest of the paper is organized as follows: Section 2 briefly presents the standard backpropagation algorithm. An introduction to learning automata is given in Sect. 3. In Sect. 4 the variable learning rate algorithm is described. Section 5 presents learning au-

Manuscript received August 20, 2001.
Manuscript revised May 31, 2002.

[†]The authors are with the Department of Electrical Engineering, Amirkabir University of Technology, Hafez Ave. 424, Tehran 15914, Iran.
[††]The author is with the Department of Computer Engineering, Amirkabir University of Technology, Hafez Ave. 424, Tehran 15914, Iran.
[*]Presently, with the School of Electrical and Computer Engineering, Department of Computer Science, Oklahoma State University, menhaj@cs-okstate.edu

tomata based methods. Section 6 introduces a new algorithm named "Adaptive Variable Learning Rate," ALVR. The simulation results are given in Sect. 7. Section 8 concludes the paper.

2. The Backpropagation Algorithm

The backpropagation (BP) is a systematic method for training multilayer neural networks. The BP algorithm has two computational phases, forward and backward phase [20].

Forward phase: this phase is described by the following equations:

$$\begin{aligned} &\underline{a}^0(k) = \underline{p}(k) \\ &\underline{a}^{l+1}(k) = \underline{F}^{l+1}\left(W^{l+1}(k)\underline{a}^l + \underline{b}^{l+1}(k)\right), \\ &\qquad\qquad l = 0, 1, \ldots, L-1 \\ &\underline{a}(k) = \underline{a}^L(k) \end{aligned} \tag{1}$$

In the above, $\underline{a}^0(k)$ is the vector of function signals of neurons in the input layer at iteration k, $\underline{p}(k)$ is the input vector at iteration k, $\underline{a}^{l+1}(k)$ is the vector of function signals of neurons in layer $l+1$ at iteration k, $\underline{F}^{l+1}(.)$ is the vector of activation functions of layer $l+1$, $W^{l+1}(k)$ is the synaptic weight matrix of layer $l+1$ at iteration k, $\underline{a}^l(k)$ is the vector of function signals of neurons in layer l at iteration k, $\underline{b}^{l+1}(k)$ is the bias vector of layer $l+1$ at iteration k, $\underline{a}(k)$ is the output vector at iteration k, $\underline{a}^L(k)$ is the vector of function signals of neurons in the output layer L at iteration k and L is number of layers. In this phase, network weights and biases all kept fixed. Activation functions act on all neurons, that is:

$$\begin{aligned} &\underline{F}^{l+1}\left(\underline{n}^{l+1}(k)\right) \\ &\quad = \left[f^{l+1}\left(n_1^{l+1}(k)\right), \ldots, f^{l+1}\left(n_{s_{l+1}}^{l+1}(k)\right)\right]^T \end{aligned} \tag{2}$$

where $\underline{n}^{l+1}(k)$ is the vector of net internal activity levels of neurons in layer $l+1$ at iteration k, $f^{l+1}(.)$ is the activation function of layer $l+1$, $n_i^{l+1}(k)$ is the net internal activity level of neuron i in layer $l+1$ at iteration k and s_{l+1} is the number of neurons in layer $l+1$.

Backward phase: in this phase, sensitivity vectors are propagated from output layer to input layer. The following equation describes dynamics of the backward phase:

$$\begin{aligned} &\underline{\delta}^L(k) = -2\dot{F}^L(\underline{n})\underline{e}(k) \\ &\underline{\delta}^l(k) = \dot{F}^l\left(\underline{n}^l(k)\right)\left(W^{l+1}(k)\right)^T\underline{\delta}^{l+1}(k), \\ &\qquad\qquad l = L-1, \ldots, 1 \\ &\underline{e}(k) = \underline{t}(k) - \underline{a}(k) \end{aligned} \tag{3}$$

where $\underline{\delta}^L(k)$ is the vector of local gradients of neurons in the output layer L at iteration k, $\dot{F}^L(.)$ is the vector derivative of activation functions of the output layer L, $\underline{e}(k)$ is the error vector at iteration k, $\underline{\delta}^l(k)$ is the local gradients of neurons in layer l at iteration k, $\dot{F}^l(.)$ is the vector of derivatives of activation functions of layer l, $\underline{\delta}^{l+1}(k)$ is the vector of local gradients of neurons in layer $l+1$ at iteration k and $\underline{t}(k)$ is the desired response vector at iteration k. In the backward phase, because of the availability of the target vector the error vector is first computed. Then, the error vector from right to left and from the output layer to the input layer are propagated and local gradients, neuron by neuron are computed using a recursive algorithm.

Parameter adjustment: In this step, weight matrices and biases are adjusted as follows.

$$\begin{aligned} &W^l(k+1) = W^l(k) - \alpha\underline{\delta}^l(k)\left(\underline{a}^{l-1}(k)\right)^T \\ &b^l(k+1) = b^l(k) - \alpha\underline{\delta}^l(k), \quad l = 1, 2, \ldots, L \end{aligned} \tag{4}$$

Where $W^l(k+1)$ is the synaptic weight matrix of layer l at iteration $k+1$, $\underline{b}^l(k+1)$ is the bias vector of layer l at iteration $k+1$ and α is the learning rate parameter.

Stopping criterion: If the average of error squares in each epoch (sum of square errors for all of training patterns) is smaller than a predetermined value or the rate of changes in network parameters after each cycle is very small, then the BP algorithm is terminated.

3. Learning Automata

Learning automata (LA) can be classified into two main groups, fixed and variable structure learning automata (FSLA and VSLA) [21]. If the transition probabilities from one state to another state and probabilities of the corresponding actions and states are fixed, the automaton is said to be fixed- structure automata, otherwise the automaton is called variable- structure automata. Examples of the FSLA type are Tsetlin, Krinsky and Krylov automata.

A fixed structure learning automaton can be defined by the quintuple $\langle\underline{\alpha}, \underline{\Phi}, \underline{\beta}, F(.,.), G(.)\rangle$ where:

(1) The output of an automaton at the instant n, denoted by $\alpha(n)$, is an element of the finite set $\underline{\alpha} = (\alpha_1, \ldots, \alpha_r)$ where r is the number of actions.
(2) The state of the automaton at any instant n, denoted by $\Phi(n)$, is an element of the finite set $\Phi = (\Phi_1, \ldots, \Phi_s)$ where s is the number of states.
(3) The input of an automaton at the instant n, denoted by $\beta(n)$, is an element of a set $\underline{\beta} = \{0, 1\}$, where 1 represents a penalty and 0 a reward.
(4) The transition function $F(.,.)$ determines the state at the instant $(n+1)$ in terms of the state and the input at the instant n:

$$\Phi(n+1) = F[\Phi(n), \beta(n)]$$

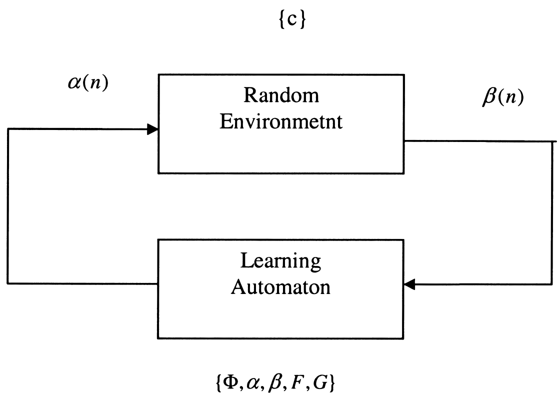

Fig. 1 The interconnection of the learning automata and environment.

$F(.,.)$ could be either deterministic or stochastic.

(5) The output function $G(.)$ determines the output of the automaton at any instant n in terms of the state at that instant:

$$\alpha(n) = G[\Phi(n)]$$

$G(.)$ could be either deterministic or stochastic.

The selected action serves as the input to the environment that 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 an action dependent quantity. The element c_i of $\underline{c} = \{c_1, c_2, \ldots, c_r\}$, which characterizes the environment, may then be defined by the following equation:

$$\Pr(\beta(n) = 1 \mid \alpha(n) = \alpha_i) = c_i \quad (i = 1, 2, \ldots, r) \tag{5}$$

Consequently, c_i represents the probability that the application of an action α_i to the environment will result in a penalty output. On the basis of the response $\beta(n)$, the state of the automaton $\Phi(n)$ is updated and a new action is then chosen at the instant $(n+1)$. Note that the c_i's are initially unknown and it is desired that, as a result of its interaction with the environment, the automaton arrives at the action leading to the minimum penalty response in an expected sense. The interconnection of the learning automaton and environment is shown in Fig. 1. In the following sections, we will describe fixed structure and variable structure learning automata.

3.1 The Two-State Automaton ($L_{2,2}$)

This automaton has two states, Φ_1 and Φ_2 and two actions α_1 and α_2. The automaton receives an input from a set of $\{0, 1\}$ and switches its states upon encountering an input 1 (unfavorable response) and remains unchanged in the same state by receiving an input 0 (favorable response). An automaton that uses this strategy is referred as $L_{2,2}$ where the first subscript refers to the number of states and second subscript represents total number of actions. State transitions are shown in Fig. 2.

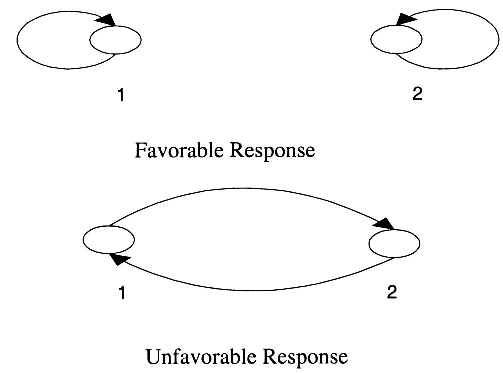

Fig. 2 The state transition graph for $L_{2,2}$.

3.2 The Two-Action Automaton with Memory

The $L_{2N,2}$ 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. $L_{2N,2}$ keeps an account of the number of successes and failures received for each action. The automaton switches from one action to another only when the number of failures exceeds the number of successes (or some maximum value N) by one.

The procedure described above is a convenient method to keep a track of the performance of the actions α_1 and α_2. Here, N is called the memory depth associated with each action, and the automaton is said to have a total memory of $2N$. For every favorable response, the state of the $L_{2N,2}$ automaton moves deeper into the memory of the corresponding action, conversely it moves out of it for an unfavorable response. This automaton can be extended to multiple actions and it is named $L_{MN,M}$ automaton where M denotes the number of actions. The state transition graph of $L_{2N,2}$ automaton is shown in Fig. 3.

3.3 The Krinsky Automaton

The Krinsky automaton behaves exactly like $L_{2N,2}$ automaton when the response of the environment is unfavorable. However, for a favorable response, any state Φ_i $(i = 1, 2, \ldots, N)$ returns to the state Φ_1 and any state Φ_i $(i = N+1, N+2, \ldots, 2N)$ moves to the state Φ_{N+1}. This, in turn, implies that a string of N consecutive unfavorable responses are needed, in general, for moving from one action to another. The state transi-

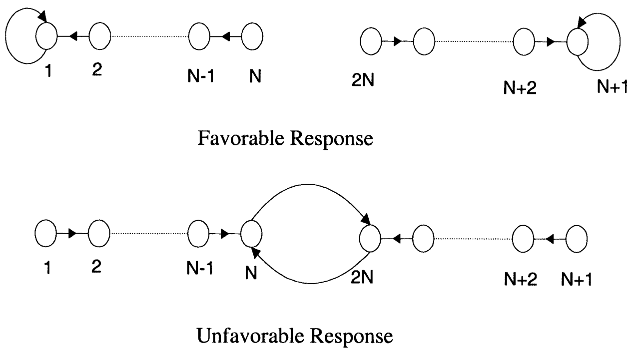

Fig. 3 The state transition graph for $L_{2N,2}$.

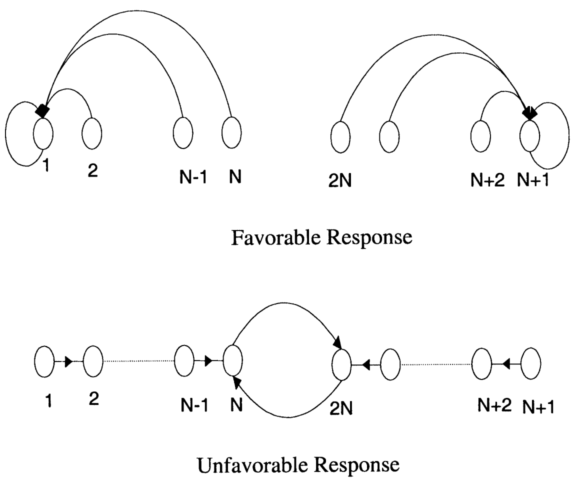

Fig. 4 The state transition graph for the Krinsky automata.

tion graph of Krinsky automaton is shown in Fig. 4.

3.4 The Krylov Automaton

This automaton has state transitions that are identical to the $L_{2N,2}$ automaton when the output of the environment is favorable. However, when the response of the environment is unfavorable, a state Φ_i ($i \neq 1, N, N+1, 2N$) goes to a state Φ_{i+1} with probability 0.5 and to a state Φ_{i-1} with a 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 $i = N$, Φ_N moves to Φ_{N-1} and Φ_{2N} each with probability 0.5 and similarly, when $i = 2N$, Φ_{2N} moves to Φ_{2N-1} and Φ_N each with probability 0.5. The state transition of Krylov automaton is shown in Fig. 5.

3.5 Variable Structure Learning Automata

Variable structure learning automaton update either the transition probabilities or the action probabilities on the basis of the input. A variable learning automaton is represented by a sextuple $\langle \underline{\beta}, \underline{\Phi}, \underline{\alpha}, \underline{P}, G, T \rangle$, where $\underline{\beta}$ is a set of inputs actions, $\underline{\Phi}$ is a set of internal

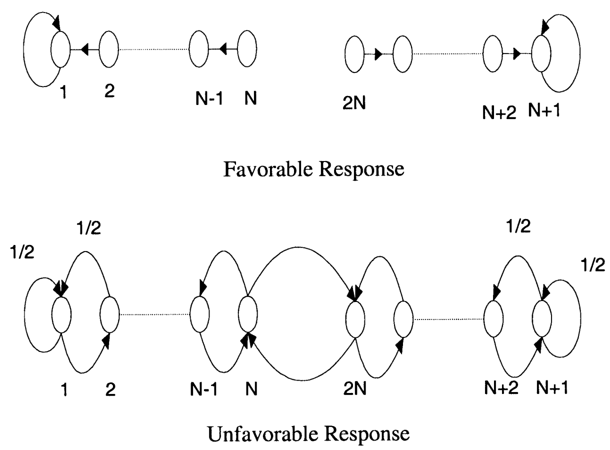

Fig. 5 The state transition graph for the Krylov automata.

states, $\underline{\alpha}$ is a set of outputs, $\underline{P}$ denotes the state probability vector determining the choice of the state at each stage k, G is the output mapping, and T denotes 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 automaton is a learning algorithm for updating the action probabilities. Various learning algorithms have been reported in the literature [21]. Let α_i be the action chosen at time k as a sample realization of the distribution $p(k)$. In linear reward-penalty (L_{R-P}) algorithm the recurrence equation for updating p is defined as:

Favorable response $\beta(n) = 0$

$$\begin{aligned} p_i(n+1) &= p_i(n) + a[1 - p_i(n)] \\ p_j(n+1) &= (1-a)p_j(n), \quad j \neq i \end{aligned} \tag{6}$$

Unfavorable response $\beta(n) = 1$

$$\begin{aligned} p_i(n+1) &= (1-b)p_i(n) \\ p_j(n+1) &= b/(r-1) + (1-b)p_j(n) \quad j \neq i \end{aligned} \tag{7}$$

Parameters a and b are called step size and determine the amount of increment (decrement) of the action probability. Another common learning algorithm is the linear reward -inaction (L_{R-I}) algorithm. In the (L_{R-I}) algorithm, for the favorable response $\beta(n) = 0$, the probability corresponding to α_i increases and the others decrease. But for an unfavorable response $\beta(n) = 1$, these probabilities do not change. A Recursive equation for adjusting P is given below.

Favorable response $\beta(n) = 0$

$$\begin{aligned} p_i(n+1) &= p_i(n) + a[1 - p_i(n)] \\ p_j(n+1) &= (1-a)p_j(n), \quad j \neq i \end{aligned} \tag{8}$$

Unfavorable response $\beta(n) = 1$

```
%******* Training Patterns **************
p= Input Patterns Matrix;t= Target Pattern Matrix; % Initialize weight matrixes and bias vectors.
W1=w1_0 ; w2=w2_0 ; b1=b1_0 ; b2=b2_0 ;
% Set training parameters.
F1='first layer function';  F2='second layer function'; me = Maximum number of epochs to train;
eg = Sum-squared error goal; lr = Initial Learning rate; mc = Momentum constant;
im = Learning rate increment coefficient;    dm = Learning rate decrement coefficient; er = Maximum error ratio;
%******** MAIN PROG. *******************
dw1 = w1*0;db1 = b1*0; dw2 = w2*0;db2 = b2*0; MC  = 0;  % set momentum constant to zero
% Forward Phase
a1 = f1(w1*p+b1); a2 = f2(w2*a1+b2); e = t-a2; SSE = sumsqr(e);
% Backward Phase
δ2 = -2. f2•(n2) .e;  δ1 =  f1•(n1) .w2. δ2;
for i=1:me
  if SSE < eg, i=i-1; break, end
 %Updating Phase
  dw1 = mc*dw1 + (1-mc)*lr*δ1*p';  [R,Q] = size(p);  db1 = mc*db1 + (1-mc)*lr*δ1*ones(Q,1);
  dw2 = mc*dw2 + (1-mc)*lr*δ2*a1';  [R,Q] = size(a1);  db2 = mc*db2 + (1-mc)*lr*δ2*ones(Q,1);
  MC = mc; New_w1 = w1 + dw1; new_b1 = b1 + db1;  New_w2 = w2 + dw2; new_b2 = b2 + db2;
 % Forward Phase
 new_a1 = f1(new_w1*p,new_b1);  new_a2 = f2(new_w2*new_a1,new_b2);  new_e = t-new_a2;  new_SSE = sumsqr(new_e);
 %Momentum & Adaptive Learning Rate Phase
  if new_SSE > SSE*er
   lr = lr * dm;    MC = 0;
  Else
    if new_SSE < SSE      lr = lr * im;  end
   w1=new_w1; b1=new_b1; a1=new_a1; w2=new_w2; b2=new_b2; a2=new_a2;    e=new_e; SSE=new_SSE;
   % Backward Phase
   δ2 = -2. f2•(n2) .e;  δ1 =  f1•(n1) .w2. δ2;
 end
end
```

Fig. 6 Variable learning rate algorithm.

$$p_j(n+1) = p_j(n), \quad 1 \leq j \leq r \tag{9}$$

In the above equations "r" represents total number of actions.

4. The Variable Learning Rate (VLR) Algorithm

With standard steepest descent, the learning rate is held constant throughout training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. If the learning rate is set too high, the algorithm may oscillate and become unstable. If the learning rate is too small, the algorithm will take too long to converge. It is not practical to determine the optimal setting for the learning rate before training, and in fact, the optimal learning rate changes during the training process as the algorithm moves across the performance surface.

The performance of the steepest descent algorithm can be improved if we allow the learning rate to being tuned during the training process. An adaptive learning rate will attempt to keep the learning step size as large as possible while keeping the learning algorithm stable. The learning rate is made responsive to the complexity of the local error surface.

An adaptive learning rate requires some changes in the training procedure used by standard BP. First, the initial network output and error are calculated. At each epoch new weights and biases are calculated using the current learning rate. New outputs and errors are then calculated. If the new error exceeds the old error by more than a predefined ratio Max_err_ratio (typically 1.04), the new weights and biases are discarded. In addition, the learning rate is decreased (typically by multiplying by lr_dec = 0.7). Otherwise, the new weights are kept accepted. If the new error is less than the old one, the learning rate is increased (typically by multiplying by lr_inc = 1.05).

This procedure increases the learning rate, but only to the extent that the network can learn without large amount of errors. Thus, a near optimal learning rate is obtained for the local terrain. When a larger learning rate results in a stable learning, the learning rate is increased. When the learning rate is too high to guarantee a decrease in the error, it gets decreased until a stable learning resumes. A Variable learning rate algorithm for a three-layer neural network is shown in Fig. 6.

5. Learning Automata Based Methods

In this section, we describe LA-based methods for adaptation of BP parameters. In these methods, neural network acts as environment. The interconnection of neural network and learning automaton is shown in Fig. 7. Different values of the BP parameters act as a set of automaton actions. In each step, an action is selected and fed to environment. The neural network uses these parameters and runs the BP algorithm N times. Then, a function of network error is compared with the corresponding one obtained on the previous iteration. This function for example can be the minimum of errors in N iterations. If we observe a decrease in the function, the neural network generates a favorable response ($\beta(n) = 0$) otherwise, the neural network produces an unfavorable response ($\beta(n) = 1$).

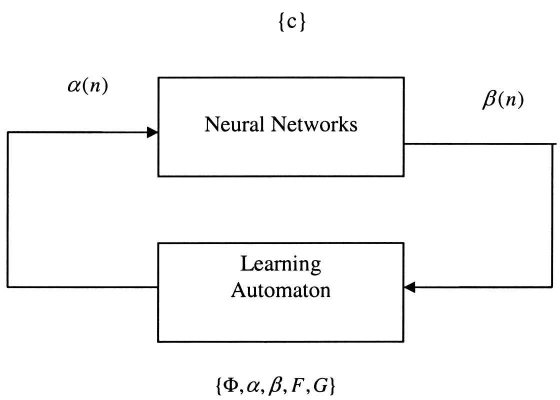

Fig. 7 The interconnection of the neural network and learning automata.

Using the network response, the learning automaton tunes the action probabilities (in the case of the LA with variable structure) or states (in the case of the LA with fixed structure). Algorithms to adapt the BP learning rate using the variable and fixed structure LAs for a three-layer neural network are shown in Figs. 8 and 9.

6. Adaptive Variable Learning Rate Algorithm

As discussed in Sect. 4, the VLR algorithm has three important parameters. These parameters are learning rate increment coefficient (lr_inc), learning rate decre-

```
%***** Training Patterns **********

p= Input Patterns Matrix; t= Target Pattern Matrix;
%Initialize weights and biases.
w1=w1_0 ; w2=w2_0 ; b1=b1_0 ; b2=b2_0 ;

%*** set training parameters *******
f1='first layer function'; f2='second layer function'; N=Number of Iteration of BP; me = Maximum number of epochs to train;
lr=learning rate; mc=Momentum constant; eg=Sum-squared error goal;
%****initialize automata parameters ********
NO_OF_ACTIONS=Number of actions;
% Initialize actions probabilities
for i=1:NO_OF_ACTIONS, pdf(i) = (1/NO_OF_ACTIONS); end
% candidates for LEARNING_RATE
LEARNING_RATE = set of actions;    λ = action prob. Increment step size ; μ = action prob. Decrement step size ;

%******* Main Prog. ******************
% select a lr from LEARNING_RATE set.
[lr, Selected_Lr_Index]=Select_Lr_VSLA();
% repeat BP algorithm, N times.
[w1,b1,w2,b2,tr]=BP()
%find min. of training error in N iteration.
Min.OfSSE=Min.(tr);
For I=1:me
  % select a Lr from LEARNING_RATE set.
  [lr, Selected_Lr_Index]=Select_Lr_VSLA();
  % repeat BP algorithm, N times.
  [w1_new,b1_new,w2_new,b2_new,tr_new]= BP()
  % find min. of train. error in N iteration.
  Min.OfSSE_new=Min.(tr_new);
  % environment responces to automatan action
  if Min.OfSSE_new > Min.OfSSE*MAX_ERR_RATIO   β=1 % penalty
  Else, β=0 % reward , w1=w1_new;w2=w2_new;b1=b1_new;b2=b2_new; Min.OfSSE = Min.OfSSE_new;, Tr = tr_new;
  End
  %automatan updates its actions' probability
  pdf=UpdateProb.(β,Selected_Lr_Index);
end
```

(a) Main program.

Fig. 8 The learning rate adaptation algorithm using the VSLA.

```
%*******Functions description ************
function[lr,Selected_Lr_Index]=Select_Lr_VSLA()
% sum elments of probability distribution fun. to compute cumulative density fun.
cdf=zeros(1, NO_OF_ACTION + 1 );

for I=1:NO_OF_ACTION,  cdf(i+1)=cdf(i)+ pdf(i);   end ; r=rand(1);
for I=1:NO_OF_ACTION
  if ( cdf(i) < r )&( r <= cdf(i+1) ), lr = LEARNING_RATE(i);;  Selected_Lr_Index=i;   End
end
function pdf = updateprob.(β, Selected_Lr_Index)
% L_{R-P} algorithm.
if β = = 0 % if Automaton receives reward response
 for I = 1:NO_OF_ACTION
  if I = = Selected_Lr_Index;  pdf(I)=pdf(I)+λ*(1-pdf(I)); else
   pdf(I)=pdf(I)*(1 – λ);; end; end
 else % if Automaton receives penalty response
 for I = 1:NO_OF_ACTION
  if I = = Selected_Lr_Index;  pdf(I)=pdf(I)*(1 – μ);; else  pdf(I)=(μ/(NO_OF_ACTION-1))+(1-μ)* pdf(I); end ; end; end
function [w1,b1,w2,b2,tr]=BP()
dw1 = w1*0; db1 = b1*0; dw2 = w2*0; db2 = b2*0;
% Forward Phase
a1 = f1(w1*p+b1); a2 = f2(w2*a1+b2); e = t-a2; SSE = sumsqr(e); Tr(1) = SSE;
for i=1:N
 % Backward Phase
 δ2 = -2. f2 •(n2) .e;   δ1 =  f1 •(n1)  .w2. δ2;
 %Updating Phase
 dw1 = mc*dw1 + (1-mc)*lr*δ1*p';
 [R,Q] = size(p); db1 = mc*db1 + (1-mc)*lr*δ1*ones(Q,1);  dw2 = mc*dw2 + (1-mc)*lr*δ2*a1';
 [R,Q] = size(a1);  db2 = mc*db2 + (1-mc)*lr*δ2*ones(Q,1);
 w1=w1+dw1; b1=b1+db1; w2=w2+dw2; b2=b2+db2;
 % Forward Phase
 a1 = f1(w1*p+b1); a2 = f2(w2*a1+b2);  e = t-a2;  SSE = sumsqr(e);  tr(i+1) = SSE;
end
```

(b) Subroutines.

Fig. 8 (Continued.)

ment coefficient (lr_dec), and the maximum error ratio (Max_err_ratio). These parameters have an important influence on the neural network learning speed of convergence. To show the influence of the VLR algorithm parameters on the learning speed, we perform a simulation on the sinusoidal function approximation problem (see Sect. 7.1). In the first experiment we choose lr_dec $= 0.7$, lr_inc $= 1.05$ and vary Max_err_ratio between 1 and 2. In the second experiment, we choose lr_dec $= 0.7$, Max_err_ratio $= 1.04$ and vary lr_inc between 1 and 2, and in the third experiment, we choose lr_inc $= 1.05$, Max_err_ratio $= 1.04$ and vary lr_dec between 0.1 and 1.

The simulation results are shown in Figs. 10 through 12. These figures show changes of the learning speed versus different parameters of the variable learning rate algorithm. As shown in these figures the learning speed is strongly dependent on these parameters. The reason behind this is as follows: It has been found when the error surface is far from the form of a quadratic bowl, it usually consists of a large amount of flat regions as well as long and narrow extremely steep regions [2]–[4]. In the simulations, first, the initial network output and the output error are calculated. At each epoch weights and biases are adjusted using the current learning rate. New outputs and errors are then calculated.

If we are in a point with a steep region, the new error exceeds the old error by more than a predefined ratio, Max_err_ratio, of typically 1.04, in this case the new weights and biases are discarded. In addition, the learning rate is decreased (typically by multiplying by lr_dec $= 0.7$). Otherwise the new weights are taken.

If the slope of the error surface is very large, we must decrease the learning rate with a high rate and if it is not very large, we can decrease the learning rate with a low rate. So, by dynamically tuning the learning rate decrement coefficient, the algorithm can get rid of high steep regions. On the other hand, if the algorithm gets trapped in a flat region, in this case the current error will be smaller than the previous error and the learning rate is incremented (multiplied by lr_inc). If the error surface is very flat, then we must increase the learning rate with a high lr_inc, while in the cases at which the error surface is not very flat, the learning rate should be increased by a low lr_inc. So, by dynamically tuning the increment coefficient, the learning algorithm will be able to escape from flat regions.

```
% Training Patterns
p= Input Patterns Matrix; t= Target Pattern Matrix;
% Initialize weights and biases.
W1=w1 0 ; w2=w2 0 ; b1=b1 0 ; b2=b2 0 ;
% set training parameters
f1='first layer function';  f2='second layer function'; N=Number of Iteration of BP; Lr=learning rate;
me = Maximum number of epochs to train; Mc=Momentum constant; eg=Sum-squared error goal;
% initialize automata parameters
NO_OF_ACTIONS=Number of actions;
% candidates for LEARNING_RATE
LEARNING_RATE = set of actions;
%******* Main Prog. ******************
% select a lr from LEARNING_RATE set.
Lr=Select_Lr_FSLA(currentstate);
% repeat BP algorithm, N times.
[w1,b1,w2,b2,tr]=BP()
% find min. of training error in N iteration.
Min.OfSSE=Min.(tr);
For I=1:me
  % select a Lr from LEARNING_RATE set.
  lr=Select_Lr_FSLA(currentstate);
  % repeat BP algorithm, N times.
  [w1_new,b1_new,w2_new,b2_new,tr_new]= BP()
  % find min. of train. error in N iteration.
  Min.OfSSE_new=Min.(tr_new);
  % environment responces to automatan action
  if Min.OfSSE_new > Min.OfSSE*MAX_ERR_RATIO
    β=1  % penalty
  Else
    β =0  % reward
    w1=w1_new;w2=w2_new;b1=b1_new;b2=b2_new;
    Min.OfSSE = Min.OfSSE_new;
    tr = tr_new;
  End
  % updates current state
  CurrentState = UpdateCurrentState(β);
End
```

(a) Main program.

```
%*******Functions description ************
function  lr = Select_Lr_FSLA(CurrentState)
for i=1:NO_OF_ACTIONS
  if((i-1)*N+1<=CurrentState))&(CurrentState<= i*N ),  lr= LEARNING_RATE(i);   end
end

Function  CurrentState = UpdateCurrentState(β)
% Tsetlin automata
if β = = 0 % reward response
 for i=1:NO_OF_ACTIONS
  if CurrentState = = (i-1)*N+1, CurrentState = CurrentState; break;
  Elseif(CurrentState >(i-1)*N+1)&( CurrentState <= i*N ), CurrentState = CurrentState - 1; break;, End, End
Else  % penalty response
 For i=1:NO_OF_ACTIONS
  If CurrentState = = i*N, if CurrentState = = NO_OF_ACTIONS*N, CurrentState = N; break;
                           Else ,  CurrentState = (i+1)*N; break; End
  Elseif(CurrentState>= (i-1)*N+1)& ( CurrentState < i*N ), CurrentState = CurrentState + 1; break; End, end
End

Function [w1,b1,w2,b2,tr]=BP()
dw1 = w1*0; db1 = b1*0; dw2 = w2*0; db2 = b2*0;
% Forward Phase
a1 = f1(w1*p+b1); a2 = f2(w2*a1+b2); e = t-a2; SSE = sumsqr(e); Tr(1) = SSE;
for i=1:N
 % Backward Phase
 δ2 = -2. f2 '(n2) .e;  δ1 =  f1 '(n1) .w2. δ2;
 % Updating Phase
 dw1 = mc*dw1 + (1-mc)*lr*δ1*p';  [R,Q] = size(p);  db1 = mc*db1 + (1-mc)*lr*δ1*ones(Q,1);
 dw2 = mc*dw2 + (1-mc)*lr*δ2*a1'; [R,Q] = size(a1);
 db2 = mc*db2 + (1-mc)*lr*δ2*ones(Q,1);
 w1=w1+dw1; b1=b1+db1; w2=w2+dw2; b2=b2+db2;
 % Forward Phase
 a1 = f1(w1*p+b1);  a2 = f2(w2*a1+b2);  e = t-a2;
  SSE = sumsqr(e); tr(i+1) = SSE;
End
```

(b) Subroutines.

Fig. 9 The learning rate adaptation algorithm using FSLA.

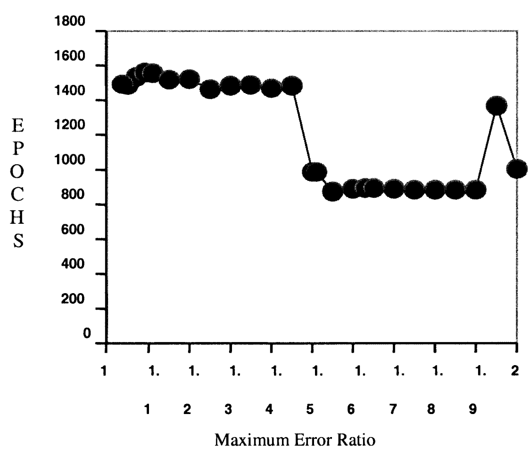

Fig. 10 The effect of the maximum error ratio.

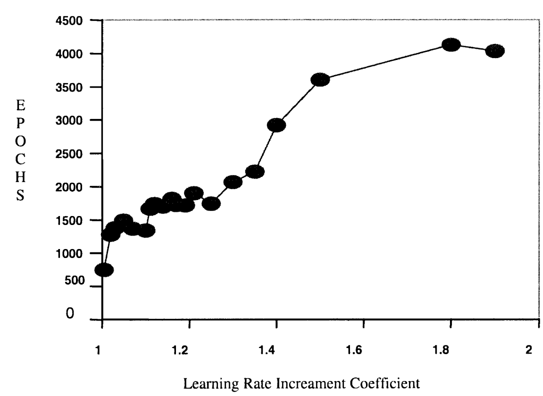

Fig. 11 The effect of the learning rate increment coefficient.

In this paper, we introduce a new algorithm named Adaptive Variable Learning Rate (AVLR). In this algorithm while keeping the main structure of variable learning rate algorithm, the algorithm parameters Max_err_ratio, Lr_Dec and Lr_Inc are tunned dynamically during the learning process. For adapting these parameters, we use the fixed structure automaton, Tsetlin.

Now the proposed AVLR algorithm is fully described. First automaton parameters are chosen. These parameters are the depth of memory, number of actions and the set of actions. In the simulation studies, we chose the depth of memory and number of actions 2, and 5, respectively and the action set as:

$$\{(0.88, 1.19, 1.043), (0.83, 1.12, 1.042),$$

$$(0.78, 1.08, 1.041), (0.7, 1.05, 1.04),$$
$$(0.63, 1.04, 1.039)\}.$$

Each action of automaton itself is a triple ordered set (i, j, k). The first component, i, is the learning rate decrement coefficient, the second one, j, is the learning rate increment coefficient and the third one, k, is the maximum error ratio. In the next step the weight matrixes and bias vectors of layers are initialized with small random numbers in $[-0.1, +0.1]$. An action is chosen from the set of actionst. Using the selected parameters from this set, the VLR algorithm is repeated STEP_SIZE times as follows.

The initial network output and error are calculated. Using the current learning rate, new weights, biases and errors are then computed. If the ratio of the new error to the old one exceeds the maximum error ratio, the new weights and biases are discarded and the learning rate is decreased proportional to the lr_dec. Otherwise, the new weights and biases are considered as valid adjustable parameters. If the new error is less than the old one, the learning rate is increased proportional to the lr_inc.

Finally, the minimum of errors in STEP_SIZE iterations is compared with the corresponding value on the previous step. If the error is decreased, the neural network rewards the learning automaton otherwise,

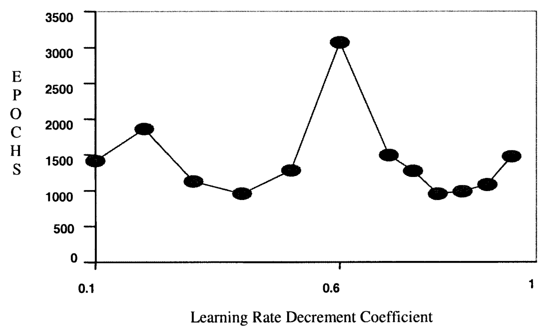

Fig. 12 The effect of the learning rate decrement coefficient.

```
% Training Patterns
p= Input Patterns Matrix; t= Target Pattern Matrix;

% initialize automata parameters
N= depth of memoryNO_OF_ACTIONS= number of actions; VLR_PARAMETERS= matrix of VLR parameters;
% Initialize weights and biases.
w1=w1 0 ; w2=w2 0 ; b1=b1 0 ; b2=b2 0 ;

% Training Parameters
f1='first layer function'; f2='second layer function'; dw1 = w1*0; db1 = b1*0; dw2 = w2*0; db2 = b2*0;
MC = 0; % set momentum constant to zero me= Maximum number of epochs to train, eg= Sum-squared error goal
lr= Learning rate, mc= Momentum constant., Im = Learning rate increment coefficient;
dm = Learning rate decrement coefficient; er = Maximum error ratio; Min.OfSSE=0; New_Min.OfSSE=1;

% ********* MAIN PROG. ************************
% Forward Phase
a1 = f1(w1*p+b1); a2 = f2(w2*a1+b2); e = t-a2; SSE = sumsqr(e);
% Backward Phase
δ2 = -2. f2 •(n2) .e; δ1 = f1 •(n1) .w2. δ2;

for i=1: me
 [im, dm, er] = Select_VLR_Param._Tsetlin( CurrentState)
 for j=1:STEP_SIZE
  if SSE < eg, j =j -1; break, end
  % LEARNING PHASE
  dw1 = mc*dw1 + (1-mc)*lr*δ1*p'; [R,Q] = size(p); db1 = mc*db1 + (1-mc)*lr*δ1*ones(Q,1);
  dw2 = mc*dw2 + (1-mc)*lr*δ2*a1'; [R,Q] = size(a1); db2 = mc*db2 + (1-mc)*lr*δ2*ones(Q,1);
  MC = mc; New_w1 = w1 + dw1; new_b1 = b1 + db1; New_w2 = w2 + dw2; new_b2 = b2 + db2;

  % PRESENTATION PHASE
  new_a1 = f1(new_w1*p + new_b1); new_a2 = f2(new_w2*new_a1 + new_b2);
  new_e = t-new_a2; new_SSE = sumsqr(new_e);

  % MOMENTUM & ADAPTIVE LEARNING RATE
  if new_SSE > SSE*er, lr = lr * dm; MC = 0;
  Else
   if new_SSE < SSE, lr = lr * im; end
   w1 = new_w1; b1 = new_b1; a1 = new_a1; w2 = new_w2; b2 = new_b2;
   a2 = new_a2; e = new_e; SSE = new_SSE;

   % Backward Phase
   δ2 = -2. f2 •(n2) .e; δ1 = f1 •(n1) .w2. δ2;
  end
 end

 % find min. of training error in STEP_SIZE iteration

 new_Min.OfSSE = Min.(tr);
 if new_Min.OfSSE >= Min.OfSSE*COEFFICIENT
  β=1 % penalize Automaton
 else
  β=0 % reward Automaton
 end
 Min.OfSSE=new_Min.OfSSE;
 CurrentState = UpdateCurrentState_Tsetlin(β)
End
```

(a) Main program.

```
%******* functions description **********
function[im,dm,er]=Select_VLR_Param._Tsetlin(CurrentState)
for I=1:NO_OF_ACTIONS
 if ( (i-1)*N +1 <= CurrentState )&( CurrentState <= i*N )
  im = VLR_PARAMETERS(i,1);
  dm = VLR_PARAMETERS(i,2);
  er = VLR_PARAMETERS(i,3);
 end
end

function CurrentState = UpdateCurrentState(β)
% Tsetlin automata
if β = = 0 % reward response
 for i=1:NO_OF_ACTIONS
  if CurrentState = = (i-1)*N+1
   CurrentState = CurrentState; break;
  Elseif (CurrentState > (i-1)*N+1 ) & ( CurrentState <= i*N )
   CurrentState = CurrentState - 1; break;
  End
 End
else % penalty response
 for i=1:NO_OF_ACTIONS
  if CurrentState = = i*N
   if CurrentState = = NO_OF_ACTIONS*N
    CurrentState = N; break;
   Else
    CurrentState = (i+1)*N; break;
   End
  elseif ( CurrentState >= (i-1)*N+1 ) & ( CurrentState < i*N )
   CurrentState = CurrentState + 1; break;
  End
 End
end
```

(b) Subroutines.

Fig. 13 The adaptive variable learning rate algorithm.

it penalizes the learning automaton. Using the neural network response, the learning automaton tunes its state. For every favorable response, the state of the Tsetlin automaton moves deeper into the memory of the corresponding action, and for an unfavorable response, it moves away from the action. This procedure is continued until a desired condition is satisfied.

Simulation results over various problems show that the proposed algorithm has a remarkable speed of convergence. The AVLR algorithm for a three-layer neural network is shown in Fig. 13.

7. Simulation Results

In this section, we compare the methods of adapting learning rate over several case study problems given below.

7.1 Sinusoidal Function Approximation

As a first problem, we use a sinusoidal function approximation.

The sinusoidal function is as follows:

$$\text{Training patterns} = \{(p_i, t_i) \mid 1 \leq i \leq 41\}$$
$$p = [-2 : 0.1 : 2], \quad t = 2\cos \pi p - 1$$

Training patterns is shown in Fig. 14. To approximate this sinusoidal function, we use a three-layer neural network with one neuron in the input layer, 5 neurons in the hidden layer and one neuron in the output layer.

7.2 Odd Parity Problem

Here, the goal is to determine whether the number of ones in input patterns is odd or even. If the number of ones is even, then the output will be one otherwise zero. In this problem, we show logical one with +1 and logical zero with −1. We choose a three-layer network with 8 neurons in the input layer, 8 neurons in the hidden layer and one neuron in the output layer. In all applications, we use the batch form for the purpose of neural network training.

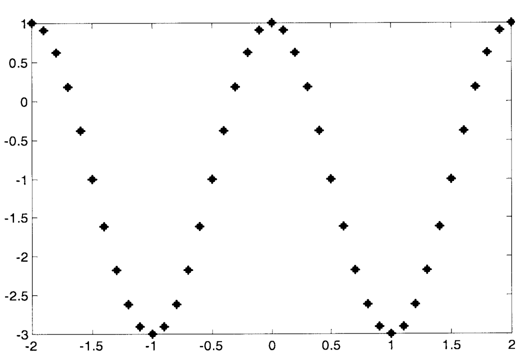

Fig. 14 Training patterns for approximating the sinusoidal function.

7.3 Symmetry Problem

In this problem, we train the neural network so that it can discriminate between symmetrical and unsymmetrical patterns. The pattern is called symmetrical if it is symmetrical with respect to the center of the pattern, otherwise it is called unsymmetrical. The network is trained so that for symmetrical patterns, the output is 1 and for unsymmetrical patterns the output becomes −1. Input patterns are 8 bits and a neural network with 8 neurons in the input layer, 2 neurons in the hidden layer and 1 neuron in the output layer is used for this problem.

7.4 Digit Recognition Problem

In this problem, we train a neural network that can recognize digits 0 to 9. For this purpose, we present numbers as shown in Fig. 15. We consider an $8 * 8$ matrix for each number. We show black blocks with 1 and white blocks with −1. Numbers 0 to 9 are coded by four bits as shown in Table 1. Therefore, the output of the network has 4 neurons. We use a three-layer neural network with 64 neurons in input layer, 6 neurons in the hidden layer and 4 neurons in the output layer.

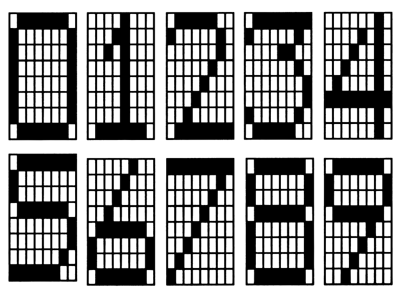

Fig. 15 Digits 0 to 9.

Table 1 Codes of digits 0 to 9.

DIGIT	BIT 3	BIT 2	BIT 1	BIT 0
0	-1	-1	-1	-1
1	-1	-1	-1	1
2	-1	-1	1	-1
3	-1	-1	1	1
4	-1	1	-1	-1
5	-1	1	-1	1
6	-1	1	1	-1
7	-1	1	1	1
8	1	-1	-1	-1
9	1	-1	-1	1

7.5 A Nonlinear System Identification Problem

Consider a second order discrete time nonlinear dynamical system described by the following equation:

$$y_{k+1} = \frac{1.5 y_k y_{k-1}}{1 + y_k^2 + y_{k-1}^2} + 0.35(y_k + y_{k-1}) + 1.2u_k \tag{10}$$

We would like to identify this system with an MLP. For this purpose, we generate inputs randomly between −1 and +1 and feed them to the network. We also choose initial conditions randomly between −1 and +1. u_k and y_k are the input and output at the instance k. We use a three-layer neural network with 3 neurons in the input layer, 8 neurons in the hidden layer and 1 neuron in the output layer.

7.6 A Phoneme Recognition Problem

In this problem the goal is to recognize Persian vowel phonemes. The speech database used in this experiment is the FarsDat database. First, voice signals are sampled with 44.1 kHz, then down sampled to 16 kHz. 14th-order LPC cepestral coefficients plus derivative of LPC cepestral coefficients and delta log energy are extracted as speech features from each frame with a length of 20 ms and a shift of 10 ms. The dimension of feature vector extracted from each frame is 25. The speech samples are pre-filtered with the filter coefficient 0.97 and Hamming windowed before calculating LPC cepestral coefficients. 938 training patterns are used in the training phase. A three-layer TDNN† neural network with a delay of 3 frames is used.

7.7 Printed Persian Letter Recognition Problem

The ten printed farsi letters are shown in Fig. 16. There is a page of 170 printed Farsi letters, 17 samples for every letter. 160 samples are used to train the network and the remaining samples are used for the testing purpose. This page is digitized with a resolution of 300 dpi. The momentum constants M_1 through M_7 are given below:

$$\begin{aligned}
M_1 &= \mu_{20} + \mu_{02} \\
M_2 &= (\mu_{20} - \mu_{02})^2 + 4(\mu_{11})^2 \\
M_3 &= (\mu_{30} - 3\mu_{12})^2 + (\mu_{21} - 3\mu_{03})^2
\end{aligned}$$

ث	ت	پ	ب	ا
د	خ	ح	چ	ج

Fig. 16 10 printed Persian letters.

$$\begin{aligned}
M_4 &= (\mu_{30} + 3\mu_{12})^2 + (\mu_{21} + 3\mu_{03})^2 \\
M_5 &= (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12}) \\
&\quad \times \left\lfloor (\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2 \right\rfloor \\
&\quad + (3\mu_{21} - 3\mu_{03})(\mu_{21} + \mu_{03}) \\
&\quad \times \left\lfloor 3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2 \right\rfloor \\
M_6 &= (\mu_{20} - 3\mu_{02}) \left\lfloor (\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2 \right\rfloor \\
&\quad + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03}) \\
M_7 &= (3\mu_{21} - \mu_{03})(\mu_{30} + 3\mu_{12}) \\
&\quad \times \left\lfloor (\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2 \right\rfloor \\
&\quad - (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03}) \\
&\quad \times \left\lfloor 3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2 \right\rfloor
\end{aligned}$$

where μ_{pq} is the scaled momentum of order $p + q$ for a letter extracted from digitized images and submitted as inputs to the network. The network used to classify these letters has 7 input nodes, 30 hidden units and 10 output units, one for each letter.

Now we present the simulation results. The methods of adapting the learning rate parameter are used for the problems listed above. In all experiments the momentum coefficient is selected as 0.98. Also, we use the batch type learning in all methods. In the following, we describe each method.

7.8 The Standard BP

In this method for all applications the learning rate is set to 0.01.

7.9 The Variable Learning Rate (VLR)

For all problems, the algorithm starts from an initial learning rate of 14, and it is dynamically tuned through the process of learning. In Fig. 17. the error and learning rate variatitions versus epochs are shown for the sinusoidal function approximation problem.

7.10 The Variable Structure Learning Automata

In this method, we use an automaton with a variable structure to adjust the learning rate. The automaton action set, which is a set of learning rates, is chosen as follows.

$$\underline{\alpha} = \{0.1, 0.094, 0.066, 0.038, 0.01\}$$

Action probability increment and decrement coefficients are 0.01 and 0.001, respectively and the Step size is 50. For an action, the BP algorithm is first repeated 50 times, then the minimum of errors in the previous step is compared with that of in the current step. If the error increases, the neural network penalizes the automaton else it rewards it.

†Time Delay Neural Network.

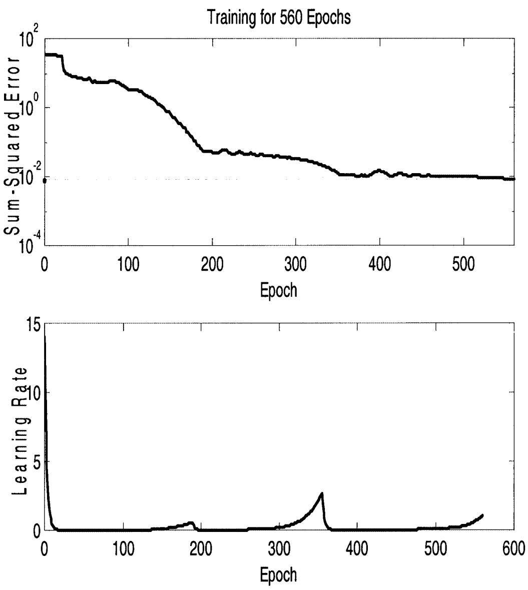

Fig. 17 The error and the learning rate variations.

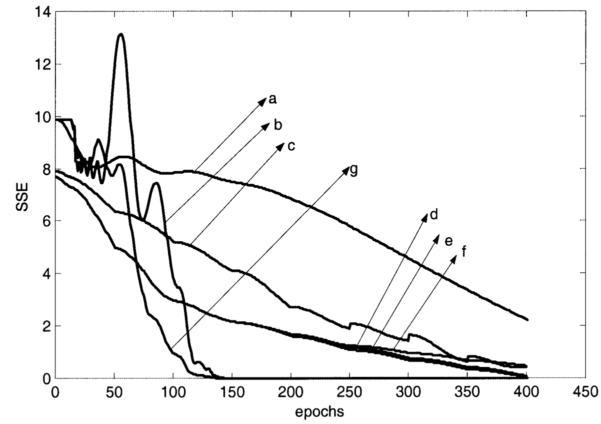

Fig. 18 The speed of of the convergence for the sinusoidal function approximation problem. (a) Standard BP, (b) VLR, (c) VSLA (5), (d) F_Tsetlin (5, 3), (e) F_Krinisky (5, 3), (f) F_Krylov (5, 3), (g) AVLR.

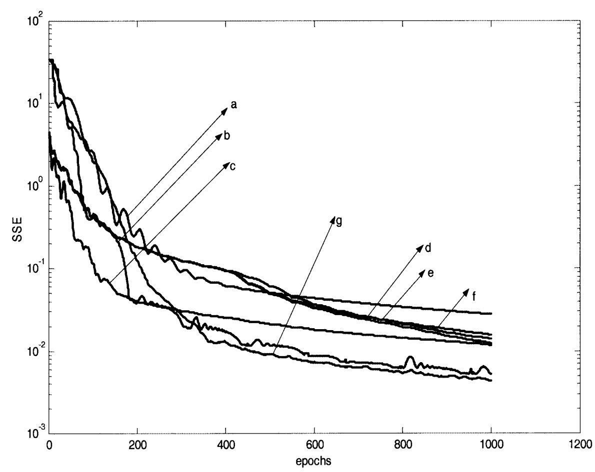

Fig. 19 The speed of the convergence for the odd parity problem. (a) Standard BP, (b) VLR, (c) VSLA (5), (d) F_Tsetline (5, 3), (e) F_Krinisky (5, 3), (f) F_Krylov (5, 3), (g) AVLR.

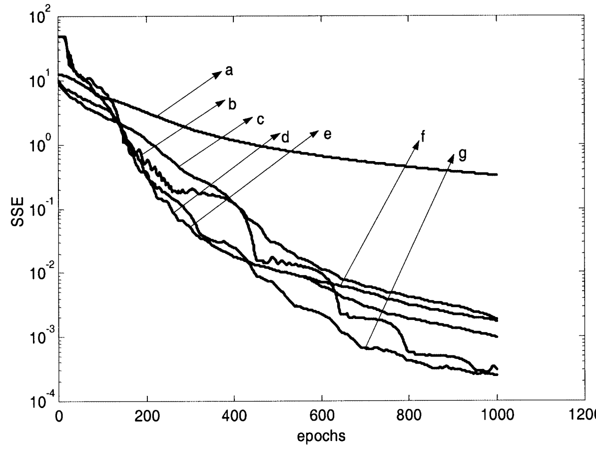

Fig. 20 The speed of the convergence for the symmetry problem. (a) Standard BP, (b) VLR, (c) VSLA (5), (d) F_Tsetlin (5, 3), (e) F_Krinisky (5, 3), (f) F_Krylov (5, 3), (g) AVLR.

7.11 The Fixed Structure Learning Automata

We use the Tsetlin, Krinsky and Krylov automata for adapting the learning rate. Similar to the variable structure automaton, we choose 5 actions as follows.

$$\underline{\alpha} = \{0.01, 0.038, 0.066, 0.094, 0.1\}$$

The depth of memory and the step size are selected 3 and 50, respectively.

7.12 The Adaptive Variable Learning Rate

In the method introduced in this paper, we use the Tsetlin automaton to tune variable learning rate parameters. We choose candidates for increment, decrement and maximum error ratio coefficients. The values that form the automaton action set are chosen as:

$$\{(0.88, 1.19, 1.043), (0.83, 1.12, 1.042), (0.78, 1.08, 1.041), (0.7, 1.05, 1.04), (0.63, 1.04, 1.039)\}.$$

The depth of memory and the step size are 2 and 50, respectively. Results of various simulations are shown in Figs. 18 through 24. In Fig. 18 various methods of adapting the learning rate for the sinusoidal function approximation problem are used. In Fig. 19, this comparison is done for the odd parity problem. Figure 20 is for the symmetry problem. Figure 21 compares these methods for the digit recognition problem. Figure 22 shows results for the second order discrete time nonlinear system identification problem. Figure 23 compares these methods for the phoneme recognition problem, and finally Fig. 24 shows the results for the

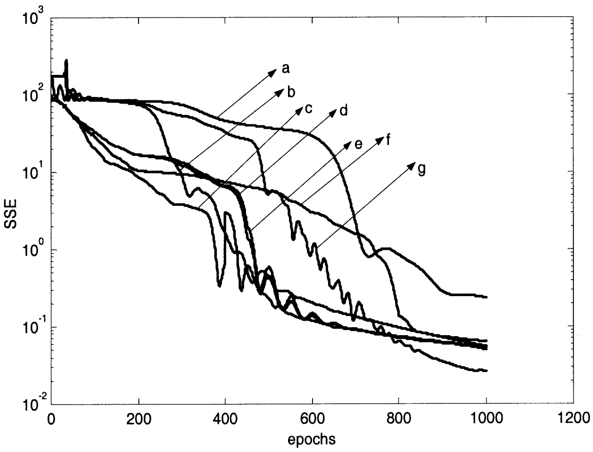

Fig. 21 The speed of the convergence for the digit recognition problem. (a) Standard BP, (b) VLR, (c) VSLA (5), (d) F_Tsetlin (5, 3), (e) F_Krinisky (5, 3), (f) F_Krylov (5, 3), (g) AVLR.

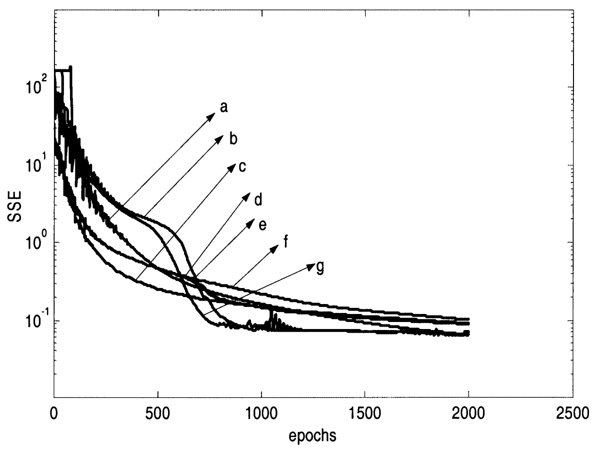

Fig. 22 The speed of convergence for the second order discrete time nonlinear system identification problem. (a) Standard BP, (b) VLR, (c) VSLA (5), (d) F_Tsetlin (5, 3), (e) F_Krinisky (5, 3), (f) F_Krylov (5, 3), (g) AVLR.

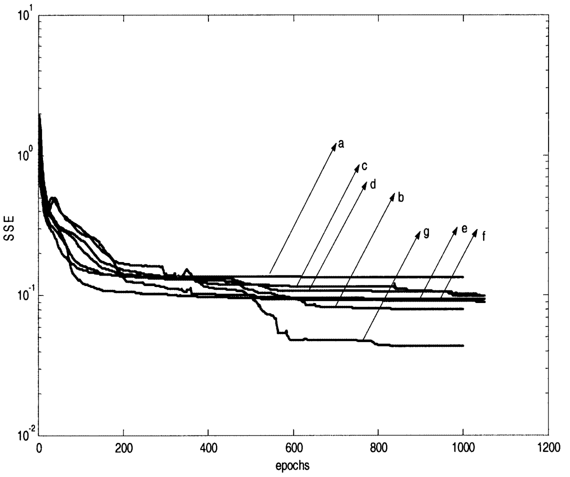

Fig. 23 Speed of convergence for the phoneme recognition problem. (a) Standard BP, (b) VLR, (c) VSLA (5), (d) F_Tsetlin (5, 3), (e) F_Krinisky (5, 3), (f) F_Krylov (5, 3), (g) AVLR.

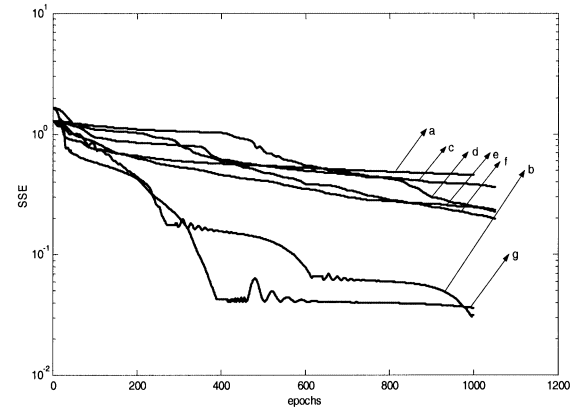

Fig. 24 The speed of convergence for the Printed Persian letter recognition problem. (a) Standard BP, (b) VLR, (c) VSLA (5), (d) F_Tsetlin (5, 3), (e) F_Krinisky (5, 3), (f) F_Krylov (5, 3) (g) AVLR.

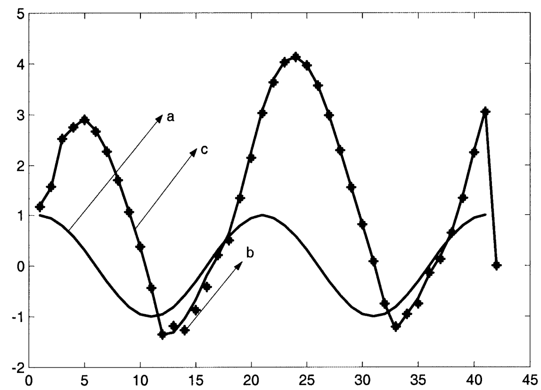

Fig. 25 (a) Input, (b) target, (c) actual output.

Persian printed letter recognition problem. As shown in these figures, the AVLR has the maximum speed and the standard BP has the minimum speed of convergence. Obviously, the AVLR outperforms the others.

Using the AVLR learning algorithm the system described in Sect. 7.5 is approximated. Here a three-layer neural network with 3 neurons in the input layer, 8 neurons in the hidden layer and 1 neuron in the output layer is employed. After 800 iterations the neural network converges with an error floor of 0.0898. For training, we use random patterns in $[-1, +1]$. To test the network's performance, we apply a sinusoidal input. Figure 18 shows the input, the target and the actual output waveforms. The target and actual output waveforms are shown with star and solid lines, respectively. As shown in Fig. 25, the output of the neural network coincides with the target waveform with an acceptable precision. For further testing of the network's performance, we double the frequency of the waveform. Figure 26 shows the results of the simulation. In this case, the output of the network tracks the target with a remarkable precision so that one can barely distinguish them.

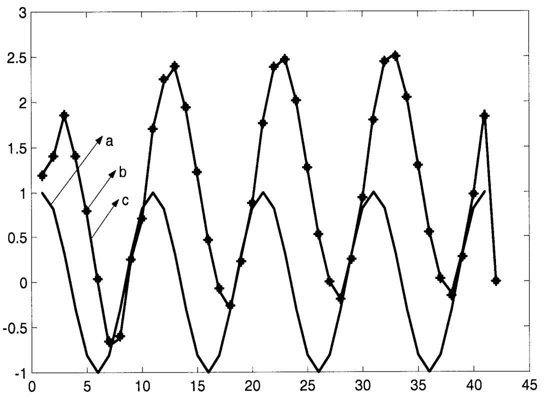

Fig. 26 (a) Input, (b) target, (c) actual output.

Table 2 The computer and the software characteristics.

CPU	PENTIUM III, 550 MHZ
Cash Memory	512 Kbyte
RAM	32 Mbyte
H.D.D.	20 Gbyte
Software	Matlab 5.3

Table 3 The time efficiency comparisons for the function approximation problem (error goal $= 10^{-3}$).

METHOD	BP	VSLA	KRYLOV	KRINISKY	TSETLIN	VLR	AVLR
Training Time (Seconds)	12.08	8.3	10.1	7.9	7.2	6.3	5.1
Number of Epochs	2919	1214	1346	1103	1086	1127	1014

7.13 Comparing Different Methods Based on Their Real Time Efficiency

To show the results on a real time scale, we conducted one simulation for each of the case study problems discussed before. The conditions of the simulations are the same as those given in the previous simulations. The computer and the software specifications are given in Table 2. The results of simulations are shown in Tables 3 through 9. In each table, the corresponding error goal is also shown. In these tables the first row represents the methods, the second row gives the training time in seconds and the third row gives the number of epochs. As shown in these tables, the training time for the standard BP is the highest and this for the proposed AVLR method is the least. By a deep look at the simulation results, we can conclude that, although in LA-based methods the selection of an action from the action set (according to actions' probability distribution in the VSLA and LA states in the FSLA), and changing the actions probabilities in the VSLA or changing states in the FSLA take a certain amount of time, the training of neural networks using LA-based

Table 4 The time efficiency comparisons for the odd parity problem (error goal $= 10^{-2}$).

METHOD	BP	VSLA	KRYLOV	KRINISKY	TSETLIN	VLR	AVLR
Training Time (Seconds)	7.08	7.076	7.00	6.90	6.8	3.14	2.94
Number of Epochs	1758	1220	1250	1232	1224	568	456

Table 5 The time efficiency comparisons for the symmetry problem (error goal $= 2 \times 10^{-3}$).

METHOD	BP	VSLA	KRYLOV	KRINISKY	TSETLIN	VLR	AVLR
Training Time (Seconds)	7.95	3.94	4.71	4.70	4.72	3.2	2.98
Number of Epochs	1947	805	1002	999	1005	703	620

Table 6 The time efficiency comparisons for the digit recognition problem (error goal $= 5 \times 10^{-2}$).

METHOD	BP	VSLA	KRYLOV	KRINISKY	TSETLIN	VLR	AVLR
Training Time (Seconds)	6.48	6.283	6.1944	6.02	6.345	5.7	5.01
Number of Epochs	1328	1030	1068	1038	1095	1000	835

Table 7 The time efficiency comparisons for the second order discrete time nonlinear system identification problem (error goal $= 10^{-1}$).

METHOD	BP	VSLA	KRYLOV	KRINISKY	TSETLIN	VLR	AVLR
Training Time (Seconds)	16.5	15.93	14.964	14.877	15	8.1	6.64
Number of Epochs	1943	1750	1720	1710	1722	940	746

Table 8 The time efficiency comparisons for the phoneme recognition problem (error goal $= 8 \times 10^{-2}$).

METHOD	BP	VSLA	KRYLOV	KRINISKY	TSETLIN	VLR	AVLR
Training Time (Seconds)	34.1	34.8	25.41	26.988	27.339	12.85	10.4
Number of Epochs	1890	1392	1303	1384	1402	670	520

Table 9 The time efficiency comparisons for the Printed Persian letter recognition problem (error goal = 10^{-1}).

METHOD	BP	VSLA	KRYLOV	KRINISKY	TSETLIN	VLR	AVLR
Training Time (Seconds)	7.6	7.3	6.5	6.02	5.92	3.02	2.86
Number of Epochs	1996	1320	1330	1304	1300	560	350

methods takes a less amount of time than the standard BP does. The reason for this is the number of epochs taken for these methods is much less than that of the standard BP. Therefore, because of the facts: 1) the time for increasing or decreasing the learning rate in the VLR method is less than that of for selection of actions and updating LA sates (in the FSLA) or actions probabilities (in the VSLA) and 2) in more cases it takes less epochs to train neural networks using the VLR method in comparison with the LA-based methods, the training time by the VLR method is less than that of the LA-based methods. And finally, because the AVLR method has a good convergence speed and it takes less epochs compared to the VLR, the training time of the AVLR becomes less than that of the VLR.

8. Conclusion

One of the most important limitations of the BP algorithm is it's low rate of convergence. To remedy this deficiency, there have been proposed several modifications to the standard BP algorithm in the literature. One of these modifications is to tune the learning rate dynamically according to gradient variations. Results of simulations show that because the VLR learning algorithm gives us a high degree of resolution freedom to choose the learning rate, it has a higher speed of convergence than any merely learning automaton based methods. In addition, because the VLR parameters have an important influence on its performance, we use the learning automaton to adjust them. In the proposed algorithm AVLR, parameters of VLR are tuned dynamically according to error variations. Simulation results on sinusoidal function approximation, odd parity, symmetry, digit recognition, second order discrete time nonlinear system identification, phoneme recognition and Persian printed letter recognition problems helped better to judge the merit of the proposed AVLR.

Acknowledgement

The authors would like to thank and appreciate the editor and reviewers for their useful comments and suggestions that helped us to enhance the quality of this work.

References

[1] D.E. Rumelhart and J.L. Mc Clelland, Distributed Processing: Explanation in the Microstructure of Cognition, vols.I, II, III, MIT Press, Cambridge, MA, 1986, 1987.
[2] B. Widrow and M.A. Lehr, "30 Years of adaptive neural networks: Perceptron, madalines, and BackPropagation," Proc. IEEE, vol.78, no.9, pp.1415–1441, 1990.
[3] D.R. Hush, J.M. Salas, and B. Horne, "Error surfaces for multilayer perceptrons," Proc. IJCNN, vol.I, pp.759–764, June 1991.
[4] X.-H. Yu, "Can back propagation error surface not have local minima," IEEE Trans. Neural Netw., vol.3, no.6, pp.1019–1021, 1992.
[5] R.A. Jacobs, "Increased rates of convergence through learning rate adaptation," Neural Networks, vol.1, no.4, pp.295–308, 1988.
[6] T.P. Vogel, J.K. Mangis, A.K. Rigler, W.T. Zink, and D.L. Alkon, "Accelarating the convergence of the backpropagation method," Biol. Cybern., vol.59, pp.257–263, 1988.
[7] R.L. Watrous, "Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization," Proc. 1st Int. Conf. Neural Networks, vol.2, pp.619–628, 1987.
[8] M.S. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," Neural Networks, vol.6, no.4, pp.525–534, 1993.
[9] A.R. Webb, D. Lowe, and M.D. Bedworth, "A Comparison of nonlinear optimization strategies for feed forward adaptive layered networks," Royal Signals and Radar Establishment, Memorandum, no.4157, July 1988.
[10] X.-H. Yu, G.-A. Chen, and S.-X. Cheng, "Dynamic learning rate optimization of the backpropagation algorithm," IEEE Trans. Neural Netw., vol.6, no.3, pp.669–677, 1995.
[11] M.B. Menhaj and M.R. Meybodi, "Application of learning automata to neural networks," Proc. Second Annual CSI Computer Conference CSIC '96, pp.209–220, Tehran, Iran, Dec. 1996.
[12] M.B. Menhaj and M.R. Meybodi, "A novel learning scheme for feedforward neural networks," Proc. ICEE-95, University of Science and Technology, Tehran, 1994.
[13] M.B. Menhaj and M.R. Meybodi, "Flexible sigmodal type functions for neural networks using game of automata," Proc. Second Annual CSI Computer Conf. CSI '96, pp.221–232, Tehran, Iran, Dec. 1996.
[14] 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.
[15] N. Baba and H. Handa, "Hierarchical structure stochastic automata can increase the efficiency of the backpropagation method with momentum," Proc. SPIE, vol.2760, pp.128–138, 1996.
[16] N. Baba and K. Sato, "A consideration on the learning algorithm of neural network-Utilization of the Hierarchical structure stochastic automata for the backpropagation method with momentum," Proc. KES '98, vol.3, pp.7–12, 1998.
[17] N. Baba and K. Sato, "Adaptive VLRBP using learning automata for neural network," Proc. WSES International Conference on: Neural Networks and Applications, 2001.
[18] N. Baba and K. Sato, "Neural networks engineering using learning automata: Introducing an adaptive algorithm for determining number of hidden Layer neurons for a three-layer neural networks," Proc. ICEE-2001, The 9th Iranian Conference on Electrical Engineering, pp.27:1–27:14, Power

and Water Institute of Technology, Tehran, Iran, 2001.

[19] M.L. Tsetlin, "On the behavior of finite automata in random media," Automatica i Telemekhanika, vol.22-10, pp.1345–1354, 1961.

[20] M.B. Menhaj, Computational Intelligence (vol.) Fundamentals of Neural Networks, Professor Hessabi Publication, 1998.

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

Behbood Mashoufi received the B.S. degree from University of Tabriz, Iran, in 1987 and M.S. degree in 1991 from Amir Kabir University of Technology, Tehran, Iran, both in electronic engineering. He is currently completing the Ph.D. degree in electronic engineering at the Amir Kabir University of Technology, Tehran, Iran, in the area of artificial neural networks and parallel processing. He is currently a Faculty Member at the electronic department of the University of Urmia, Iran. His current research interests are artificial neural networks, parallel processing and speech recognition.

Mohammad Bagher Menhaj born in Iran, received the PhD degree in electrical engineering from OSU, USA, 1992. After completing one year with OSU at postdoctoral fellow, in 1993, he joined the Amirkabir University of Technology in Tehran-Iran, where he is currently Associate Professor of control group. December 2000 to Aug. 2002, he was with school of electrical and computer engineering at OSU as visiting faculty member and research scholar. He is currently with Computer Science department as visiting faculty member and research scholar, and with school of electrical and computer engineering as research scholar. He is author and co-author of more than 170 technical papers. He is author of three books: Computational Intelligence (vol.1): Fundamentals of Neural Networks, (Professor Hesabi Center of Publishing, Tehran, 1998 and AmirKabir University, Tehran, 2000) and application of Computational Intelligence in Control (vol.1), (Professor Hesabi Center of Publishing, Tehran, 1998), all in Persian. His main research interests include theory of computational intelligence, adaptive filtering and systems, signal processing and their applications in control, power systems and communications.

Sayed A. Motamedi biography is not available.

Mohammad R. Meybodi received the B.S. and M.S. degrees in Economics from national University of Iran, in 1973 and 1977, respectively. He also received the M.S. and Ph.d. degree from Oklahama University, U.S.A., in 1980, 1983 in Computer science. Currently he is an associate prof. In computer Engineering Department, Amirkabir University of Technoligy. Prior to current position, he worked from 1983–1985 as an assistant prof. at western Michigan University, U.S.A. and from 1985–1991 as an associate prof. at Ohio University, U.S.A. His research interests include, theoretical aspects of learning systems and its applications, parallel algorithms, special purpose parallel architectures, and software development. He has published over 80 papers on international conferences and journals.