IUST-International Journal of Engineering Science, Vol. 15, No. 3, 2004, pp. 1-16
Iran University of Science and Technology.

ADAPTIVE SURVIVAL ALGORITHM

B. Mashoufi[1], M. R. Meybodi[2], S. A. Motamedi[3] and M. B. Menhaj[3]

[1] College of Electrical Engineering,
Urmia University
[2] College of Computer Engineering,
Amirkabir University of Technology
[3] College of Electrical Engineering,
Amirkabir University of Technology
Tehran, Iran
E-mail: b. mashoufi@urmia.ac.ir

Abstract: Backpropagation algorithm performs gradient descent only in the weight space of a network with fixed topology. A very small network cannot learn the problem well, and a very larger network will lead to overfitting and poor generalization performance. Algorithms that can find an appropriate network architecture automatically are thus highly desirable. The algorithms that are introduced by researchers can be classified into five major groups. Pruning algorithms, constructive algorithms, hybrid algorithms, evolutionary algorithms, and learning automata based algorithms. Meybodi and Beigy introduced the first learning automata based algorithms, called survival algorithm. This algorithm produces networks with low complexity and high generalization. Survival algorithm by turning off and on the weights, tries to find the most important weights. At the beginning, all weights of the network are on and contribute to learning. The on weights, whose absolute values are less than a threshold value, are penalized and those, whose absolute value are larger than another threshold value, are rewarded. The on weights, whose absolute values lie between these two threshold values, neither rewarded, nor penalized. The values of these two thresholds are determinative and have considerable effect on the performance of the survival algorithm. Determination of the values of these thresholds is not an easy task and usually is determined by trial and error or using past experience. In this paper, we propose a method for adaptation of these two threshold values. The proposed method have been tested on number of problems and shown through simulations that the network generated by the survival algorithm when threshold values are adapted has lesser number of weights and neurons, comparing to the network generated by the first version of the algorithm reported earlier. Experimentation shows that the adaptive survival algorithm has nearly the same degree of generalization as the non-adaptive version.

Keywords: Neural Networks Engineering, Multilayer Neural Networks, Neural Networks Topology, Backpropagation, Learning Automata.

1. INTRODUCTION

In recent years, many neural-network models have been proposed for pattern classification, function approximation and speech recognition. Among them, the class of multilayer feed forward network is perhaps the most popular. Methods using standard backpropagation, perform gradient descent only in the weight space of a network with fixed topology [1]. In general, this approach is useful only when the network architecture is chosen correctly. Too small a network cannot learn the problem well, but too large a size will lead to overfitting and poor generalization performance. This can be easily understood by analogy to the problem of curve fitting using polynomials. Consider a data set generated from a smooth underlying function with additive noise on the outputs. A polynomial with too few coefficients will be unable to capture the underlying function from which the data was generated, while a polynomial with too many coefficients will fit the noise in the data and again result in a poor representation of the underlying function. For an optimal number of coefficients, the fitted polynomial will give the best representation of the function and also the best prediction for new data.

A similar situation arises in the application of neural networks, where it is again necessary to match the complexity of the model to the problem being solved. Algorithms that can find appropriate network architecture automatically are thus highly desirable. The algorithms reported in the literatures can be classified into five major groups [46].

1.1. Pruning algorithms: pruning algorithms start with a larger network and excises unnecessary weights and/or neurons. This approach combines the advantages of training large networks (i. e., learning speed and avoidance of local minima) and higher generalization ability [3, 6, 7, 14].

1.2. Constructive algorithms: These algorithms start with a small initial network nd gradually adds new hidden units or layers until learning take places [1, 15, 19].

1.3. Hybrid algorithms: Hybrid algorithms are combination of pruning and constructive algorithms. These algorithms try to attain a satisfactory solution by adding or removing hidden units and weights [20, 21].

1.4. Evolutionary algorithms: These algorithms use a performance index such as minimum error, and search in parameter space for optimal structure. In these algorithms, each point of the search space corresponds to a network structure [22, 26].

1.5. LA based algorithms: Maybodi and Beigy introduced the first learning automata based algorithm, called survival algorithm, for obtaining optimal structures [2, 27]. In this algorithm an object migrating learning automata is used as a general search tool. This algorithm determines optimal structure with low learning complexity and high generalization ability for three layer neural network. Survival algorithm starts with a large three layer neural network, and by adding and removing hidden neurons determines number of neurons of hidden layer. Meybodi and Beigy presented three other object migrating learning automata based algorithms [46]. These algorithms consist of two parts. In the first part the required number of hidden units, and in the second part the number of weights of each neuron is determined. In the first algorithm, survival algorithm is used for determining the number of hidden units. For determining the number of weights, again an object migrating learning automata is used. This algorithm by removing the weights which have less effect, not only reduces the network complexity but also increases network generalization ability. Weight determination algorithm by turning off and on the weights, tries to find the most appropriate weights. In the other two algorithms, the goal is only, the determination of the number of weights of hidden layer. The number of units of hidden layer is determined implicitly by determining the number of weights. In the weight reduction algorithm, [46] for determining weights of hidden layer, an object migrating automaton is used. This automaton has two actions each with depth of N. Weight set $W= \{w_1, w_2, w_3, ...\}$ are placed on automata states. At the beginning, all weights of the network are on, and contribute to learning for a predetermined period. After this period is over, weights are inspected. The on weights, whose absolute values are less than a threshold value, are penalized and those, which are larger than another threshold value, are rewarded. The on weights whose absolute values lie between these two threshold values, neither rewarded, nor penalized. This way the weights which are the most important are determined. The values of these two thresholds are very determinative and have considerable effect on the performance of the algorithm. In this paper, we propose an algorithm for adaptation of these two threshold values. This algorithm is again based on learning automata. The proposed algorithm have been tested on number of problems and shown through simulations that the networks generated have lesser number of weights and neuron comparing to the networks generated by non-adaptive version of the algorithm reported by Meybodi and Beigy. It has been noticed that proposed algorithm has the same degree of generalization as the non-adaptive version.

The rest of the paper is organized as follows: Section 2 briefly presents the standard backpropagation algorithm. Overtraining and generalization is described in section 3. An introduction to learning automata is given in section 4. Proposed algorithm is described in section 5. The simulation results are given in section 6 and section 7 concludes the paper.

2. BACKPROPAGATION ALGORITHM

BP is a systematic method for training multilayer neural networks. BP algorithm has two computational phases: forward phase and backward phase [20].

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

$$\underline{a}^0 = \underline{p}(k)$$

$$\underline{a}^{l+1}(k) = \underline{F}^{l+1}(W^{l+1}(k)\,\underline{a}^l + \underline{b}^{l+1}(k)), \quad l = 0,1,...,L-1$$

$$\underline{a} = \underline{a}^L(k)$$

Activation function act on all neurons, that is:

$$\underline{F}^{l+1}(\underline{n}(k)) = \left[f^{l+1}(n_1(k)...f^{l+1}(n_{s_{l+1}}(k))\right]^T$$

2.2. Backward phase: in this phase, sensitivity vectors are propagated from output layer to input layer.

The following equations describe dynamic of the backward phase:

$$\underline{\delta}^L(k) = -2\dot{F}^L(n)\,\underline{e}(k)$$

$$\underline{\delta}^l(k) = \dot{F}^l(n^l)(W^{l+1})^T\,\underline{\delta}^{l+1}, \quad l = L-1,...,1$$

$$\underline{e}(k) = \underline{t}(k) - \underline{a}(k)$$

In the backward phase first, error vector is computed. Then, the error vector from right to left and from output layer to input layer is propagated and local gradients, neuron by neuron are computed.

2.3. Parameter adjusting: in this step weight matrices and biases are adjusted as follows:

$$W^l(k+1) = W^l(k) - \alpha\,\underline{\delta}^l(k)\left(\underline{a}^{l-1}(k)\right)^T$$

$$b^l(k+1) = b^l(k) - \alpha\,\underline{\delta}^l(k), \quad l = 1,2,...,L$$

2.4. Stopping criterion: if average of error squares in each epoch (sum of square error for all of the training patterns) is smaller than a predetermined value then BP algorithm stops.

3. OVERTRAINING AND GENERALIZATION

When a neural network is trained, the weights are modified in order to decrease the error on the training patterns. If the network is tested on a slightly different set of the same task, the error on the test set tends to decrease in step with the training error as the network generalizes from the training data to the underlying function. Fig. 1 illustrates the situation schematically. If the training data is incomplete, however, it may contain spurious and misleading regularities due to sampling [45].

At some points, usually in the later stages of learning, the network starts to take advantages of these idiosyncrasies in the training data and the error starts to increase again even though the training error continues to decrease [29].

One approach to avoid overfitting is to estimate the generalization ability during training and stop when it begins to decrease. The simplest method is to divide the data into a training set and a validation set, as above. The training set is used to modify the weights, the validation set is used to estimate the generalization ability, and training is stopped when the error on the validation set begins to rise. This technique, however, may not be practical when only a small amount of data is available, since the validation data cannot be used for training [30]-[32].

Another way of avoiding overtraining is to limit the ability of the network to take advantage of spurious correlation in the data. Overfitting is thought to happen when the network has more degrees of freedom (The number of weights, roughly) than the number of the training samples-when there are not enough examples to constrain the network. Even though it may give exactly the right output at the training points, it may be very inaccurate at other points.

Small networks beside high generalization ability have other advantages. Usually these can be implemented easily and with low cost. Theories and formulas are used for estimating required size of the system [33]-[35]. This methods correlate complexity of learning system with number of required patterns for learning a function. If number of patterns is small with respect to the system complexity we expect that the generalization error are large.

Figure 2 and 3 show pruning effect. Figure 2 shows the boundary formed by an intentionally

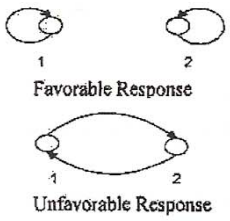


Fig. 5. The state transition graph for $L_{2,2}$

4.2. The two-action automaton with memory

This automata has 2N states and two actions and attempts to incorporate the past behavior of the system in its decision rule for choosing the sequence of actions. While the automata $L_{2N,2}$ keeps an account of the number of success and failures received for each action. It is only when the number of failures exceeds the number of successes, or some maximum value N, the automata switches from one action to the another. For every favorable response, the state of automata moves deeper into the memory of corresponding action, and for an unfavorable response, moves out of it. The state transition graph of $L_{2N,2}$ automata is shown in figure 6.

4.3. The Krinsky automaton

This automata behaves exactly like $L_{2N,2}$ automata when the response of the environment is unfavorable, but for favorable response, any state Φ_i (for i=1 … N) passes to the state Φ_1 and any state Φ_i (for i=N+1 … 2N) passes to the state Φ_{N+1}. The state transition graph of Krinsky automaton is shown in figure 7.

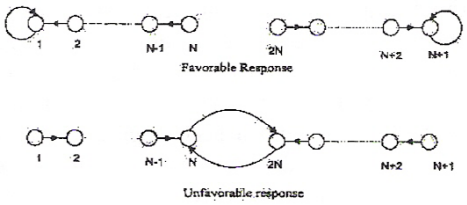


Fig. 6. The state transition graph for $L_{2N,2}$

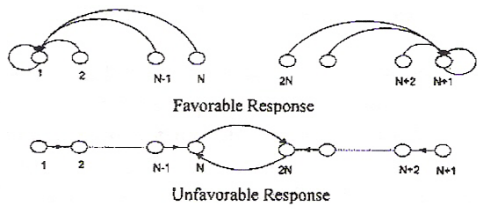


Fig. 7. The state transition graph for Krinsky Automata

4.4. The Krylov automaton

This automaton has state transition that is identical to the $L_{2,2}$ automata when the output of the environment is favorable.

However, when the response of the environment is unfavorable, a state Φ_i (i≠1, N, N+1 and 2N) passes to a state Φ_{i+1} with probability 0.5 and to a state Φ_{i-1} with probability 0.5. When i=1 or i=N+1, Φ_i stays in the same state with probability 0.5 and moves to Φ_{i+1} with the same probability. When i=N, automata state moves Φ_{N-1} to Φ_{2N} and with the same probability 0.5. When i=2N, automata state moves Φ_{2N-1} to Φ_N with the same probability 0.5. The state transition of Krylov automata is shown in figure 8.

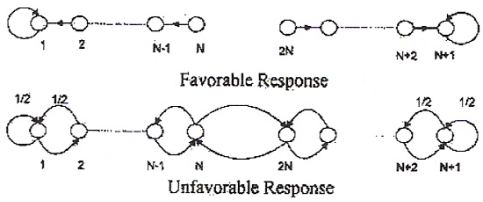


Fig. 8. The state transition graph for Krylov Automata

4.5. Variable structure learning automata

This automata is represented by sextuple $<\underline{\beta},\underline{\Phi},\underline{\alpha},\underline{P},G,T>$, where $\underline{\beta}$ is a set of inputs actions, $\underline{\Phi}$ is a set of internal states, $\underline{\alpha}$ 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 is learning algorithm. The learning algorithm is a recurrence relation and is used to modify the state probability vector. Various learning algorithms have been reported in the literature [16]. Let α_i be the action chosen at time k as a sample realization from distribution p(k). In linear reward-penalty (L_{R-P}) algorithm the recurrence equation for updating p is defined as:

Favorable response $\beta(n)=0$

$$P_i(n+1)=P_i(n)+a[1-P_i(n)]$$
$$P_j(n+1)=(1-a)P_j(n) \qquad j\neq i$$

Unfavorable response $\beta(n)=1$

$$p_i(n+1) = (1-b)p_i(n)$$
$$p_j(n+1) = b/(r-1) + (1-b)p_j(n) \qquad j \neq i$$

Parameters a and b are called step size and determine amount of increment (decrement) of the action probability. Another learning algorithm that usually used is linear reward-inaction (L_{R-I}) algorithm. In (L_{R-I}) algorithm for favorable response $\beta(n) = 0$, probability corresponding to α_i increase and others decrease. But for unfavorable response $\beta(n) = 1$ probabilities are not changed. Recursive equation for changing p is as follows:

Favorable response $\beta(n) = 0$

$$p_i(n+1) = p_i(n) + a[1 - p_i(n)]$$
$$p_j(n+1) = (1-a)p_j(n) \qquad j \neq i$$

Favorable response $\beta(n) = 1$

$$p_j(n+1) = p_j(n) \qquad 1 \leq j \leq r$$

In the above equations r represents number of action.

4.6. Object Migrating Automata

Object Migrating Automata [46] is shown by quintuple $\langle a, \Phi, \beta, F, G \rangle$ [40].

Where $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_r\}$ is action set, $\Phi = \{\Phi_1, \Phi_2, ..., \Phi_r\}$ is set of states, $\beta\{0,1\}$ is set of input, $F : \Phi \times \beta \rightarrow \Phi$ is states map function and $G : \Phi \rightarrow \alpha$ is automaton output map function. This automaton are used for object classification [40], [41], graph partitioning [42], graph correspondence [43] and determining number of hidden units of neural networks [44]. In this automaton each action show one set of patterns. In automaton with fixed structure, response of environment to automaton changes state of automaton. While in object migration automaton, objects are associated to automaton states and response of environment to automaton causes circulation of object between states of automaton. This circulation does object classification. If object W_i lies in action number j, this object belongs to group number j. For action α_k state set are considered $\{\Phi_{(k-1)N+1}, ..., \Phi_{kN}\}$.

Where N shows depth of memory. Without loss of generality $\Phi_{(k-1)N+1}$ is the most inner state and Φ_{kN} is the most outer state. If object W_i and W_j are in states $\Phi_{(k-1)N+1}$ and $\Phi_{(k-1)N+m}$ respective-ly, then the membership probability of that W_i is larger than W_j. Therefore for action α_k, the state $\Phi_{(k-1)N+1}$ is called the state with highest probability and the state Φ_{kN} is called the state with lowest probability.

5. PROPOSED ALGORITHM

In this section at first, we briefly describe the weight determination algorithm, then the threshold adaptation algorithm is presented.

5.1. Weight determination algorithm: In this algorithm an object migrating automata is used to determine the most important weights. This automaton whose state transition is shown in figure 9 has two actions. At the beginning all weights $W=\{w_1, w_2, w_3, ...\}$ are on and placed on state Φ_1. All on weights contribute to training for a predetermined period. After this period is over, weights are inspected, and those whose degree of importance are not high are penalized, those whose degree of importance is high are rewarded, and those about which we are indecisive, neither punished nor rewarded. To differentiate between a node with high degree of importance and low degree of improtance the following two heuristics are used. A weight has a high degree of importance if its input signal hasn't been attenuated considerably, or in other words its absolute value isn't small. A weight has a low degree of importance if its input signal has been attenuated considerably, or in other words is absolute value is small.

The on weights whose absolute values are less than a threshold value, are penalized and those whose absolute values are larger than another threshold value, are rewarded. The on weights, whose absolute values lie between these two threshold values, neither rewarded, nor penalized.

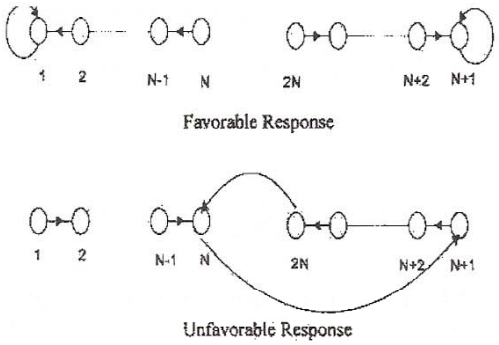


Fig. 9. Automaton HWLA(2, N, 1)

5.2. How to discriminate among the on weights:
We repeat backpropagation algorithm for a given number of times, then, on weights whose absolute values are less than a threshold value, are not considered appropriate and are penalized and those whose absolute values are larger than another threshold value, are considered inappropriate and are rewarded. The on weights, Whose absolute values lie between these two threshold values, neither rewarded, nor penalized. To determine the threshold values, the variance of absolute value of weights is used. The variance of weights for each neuron is computed as follows:

$$\sigma_h = \sqrt{\frac{\sum_{w_{hk} \in ON_w(h)} (|W_{hk}| - \mu_{ON}(h))^2}{|ON_W(h)|}} \qquad h \in ON$$

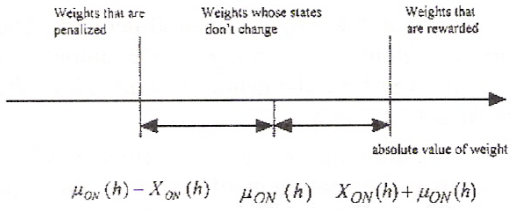


Fig. 10

Where ON is set of on units, $ON_W(h)$ is set of input on weights connected to hidden unit h and $\mu_W(h)$ is the average of absolute value of weights connected to hidden unit h. $X_{ON}(h)$ is on width (figure 10) and is computed as follows.

$$X_{ON}(h) = \lambda_{ON} \frac{OFF_W(h)}{|\#INPUT + 1|} \sigma_h$$

Where constant $\lambda_{ON} > 0$ is on width coefficient, #INPUT is the number of input layer and $OFF_W(h)$ is set of off weights connected to on hidden unit h.

5.3. How to discriminate among the off weights:
Since the off weights do not participate in training, when an off weight becomes on, the value used for this weight is the value that it had at the last time that it was on. If a weight is off for a long period of time its importance and hence its value is reduced. Therefore the off weight value is computed as follows:

$$W_{hk}(n+1) = W_{hk}(n)\,\lambda_d$$

The constant $0 < \lambda_d < 1$ is called weight reduction coefficient. Off weights variance is computed as follows:

$$\sigma_h = \sqrt{\frac{\sum_{w_{hk} \in OFF_w(h)} (|W_{hk}| - \mu_{OFF}(h))^2}{|OFF_W(h)|}} \qquad h \in ON$$

Where ON is on units set and $\mu_{OFF}(h)$ is average of absolute value of off weights connected to on hidden unit h. Off width, $X_{OFF}(h)$ is calculated as follows:

$$X_{OFF}(h) = \lambda_{OFF} \frac{ON_W(h)}{|\#INPUT + 1|} \sigma_h$$

Where constant $0 < \lambda_{OFF} < 1$ is off width coefficient, #INPUT is number of input layers and $ON_W(h)$ is set of on weights connected to on hidden unit h. Weights whose absolute values are smaller than $\mu_{OFF}(h) - X_{OFF}(h)$ are rewarded, those weights whose absolute values are larger than $\mu_{OFF}(h) + X_{OFF}(h)$ are penalized and those whose values lie between $[\mu_{OFF}(h) - X_{OFF}(h), \mu_{OFF}(h) + X_{OFF}(h)]$ remain unchanged (fig. 11).

5.4. Adaptive weight determination algorithm: The range $X_{ON}(h)$ has a large effect on the performance of weight determination algorithm. The on weights, whose absolute values are smaller than $\mu_{ON}(h) - X_{ON}(h)$ are penalized and on weights whose absolute values are larger than $\mu_{ON}(h) + X_{ON}(h)$ are rewarded. If $X_{on}(h)$ is small, more weights are penalized and as a result the rate of pruning the weights may increase and the algorithm produces a network that cannot learn the training patterns. If $X_{ON}(h)$ value is chosen to be very large, penalizing condition will be very severe and therefore on weights, which may not be important will not be turned off.

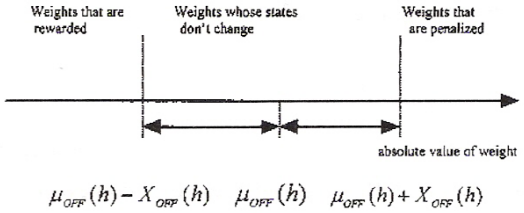


Fig. 11

$X_{ON}(h)$ is a function of several parameters as follows:

$$X_{ON}(h) = \lambda_{ON} \frac{OFF_W(h)}{|\#INPUT + 1|} \sigma_{ON}(h)$$

#INPUT is input space dimension and for a specific problem is constant. $OFF_W(h)$ is the number of off weights, and acts as inhibition factor against more weight pruning. As the number of off weights increases $X_{ON}(h)$ increases and as a result the rate of turning off the weights reduces. λ_{ON} is a constant and is called on width coefficient. $\sigma_{ON}(h)$ is the variance of on weights of hidden neuron h.

A similar discussion can be made about $X_{OFF}(h)$. The range $X_{OFF}(h)$ also has a large effect on weight determination algorithm. The off weights whose absolute value is smaller than $\mu_{OFF}(h) - X_{OFF}(h)$ are rewarded and off weights whose absolute values are larger than $\mu_{OFF}(h) + X_{OFF}(h)$ are penalized. If $X_{OFF}(h)$ is small, more off weights are penalized and as a result the rate of turning on the weights may increase and the algorithm may produce a network with high complexity and low generalization. If $X_{OFF}(h)$ value is chosen to be very large, off weight penalizing condition will be very severe and therefore off weights that need to be turned on will not be turned on. $X_{OFF}(h)$ is defined as follows:

$$X_{OFF}(h) = \lambda_{OFF} \frac{ON_W(h)}{|\#INPUT + 1|} \sigma_{OFF}(h)$$

#INPUT is input space dimension, which is constant for a given problem. $ON_w(h)$ is the number of on weights which acts as an inhibition factor against turning on more weight. λ_{OFF} is a constant which is called off width coefficient. $\sigma_{ON}(h)$ is the variance the weights of off hidden neuron h.

As we described before, weight pruning rate is a function of $X_{ON}(h)$ and weight returning rate is a function of $X_{OFF}(h)$ and by changing them we can control weight pruning and returning rate. Because $X_{ON}(h)$ and $X_{OFF}(h)$ are proportional to λ_{ON} and λ_{OFF}, so we can use these coefficients for tuning up $X_{ON}(h)$ and $X_{OFF}(h)$.

It is not easy to choose appropriate value for these coefficients for a particular application. These coefficients are usually determined by trail and error and using past experience. By having an optimum value for these parameters, we can obtain a network with low complexity and high generalization.

In this paper, by interconnection of learning automata and the survival algorithm we apply the learning automata for adapting coefficients λ_{ON} and λ_{OFF}. Input to the automata is some function of the error in the output of neural network. The learning automata according to the amount of error received from the neural network adjust the coefficients λ_{ON} and λ_{OFF}.

If the minimum of error of current epoch is larger than the minimum off error of previous epoch then the automata are penalized else rewarded. The actions of the automata correspond to the values of the coefficients.

Two Tsetline automata are used for adaptation of parameters λ_{OFF} and λ_{ON}. The action set for both automata has four values. The depth of memory is chosen to be 5.

The proposed method is summarized in the algorithm shown in Figure 12.

```
Input:
Training Patterns (p,t), p is input and t is desired output
No. of Hidden Units H
Output:
Network Weight Vector:W
Network Topology
Initialize Weights and biases
Initialize Automata Parameters
Initialize Learning Parameters
for i=1:100
    for i=1:K
     Call BP
    end
    MinimumOfError_new=minimum(Error Vector);
    Select an Action from on_width_coefficient set
    Select an Action from off_width_coefficient set
    for all Hidden Neurons
     for all weights
          if weight is OFF
              W = W*WEIGHT_DEC._COEFF.;
          end
     end
     Compute Xon
     Compute Xoff
     for all weights
          if weight is OFF
               if abs(W) < Average_off- Xoff
                    Reward OFFWeight
               elseif abs(W) > Average_off + Xoff
                         Penalize OFFWeight
               end
          if weight is ON.
               if abs(W) < Average_on - Xon
                    Penalize ON Weights
               elseif abs(W) > Average_on + Xon
                    Reward ON Weights
               end
          end
     end
    end
    if  MinOfErr_new / MinOfErr > MAX_ERR_RATIO
     EnvironmentResponce=1;
    else
     EnvironmentResponce=0;
    end
    Update_Tsetlin_State_λON (EnvironmentResponce)
    Update_Tsetlin_State_λOFF (EnvironmentResponce)
end
```

Fig. 12.

6. SIMULATION

In order to evaluate the performance of the proposed algorithm for adaptation of λ_{ON} and λ_{OFF}, simulations are carried out on four problems: English digit recognition problem, Persian printed number recognition problem, second order discrete time nonlinear function approximation problem and phoneme recognition problem. In this section we first describe the test problems and then present the results of simulations. Two sets of simulation have been conducted for each problem, one for survival algorithm and one for adaptive survival algorithm.

6.1. English Digit Recognition Problem:

There are numbers 0 to 9, and each represented by a 8*8 grid of black and white dot as shown in figure 13. We used four bits for coding numbers 0 to 9 (Table 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 to 4 output units.

6.2. Second order discrete time nonlinear function approximation problem: The goal is to approximate the following second order discrete time nonlinear function:

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

with reasonable error using a three layer network with 3 neurons in input layer, 8 neurons in hidden layer and 1 neuron in output layer. In the above equation u_k and y_k are input and output at instance k and y_{k-1} and y_{k+1} are outputs at instance k-1 and k+1. The input to the network is random numbers between –1 and +1, and the initial conditions are also chosen randomly to be between –1 and +1.

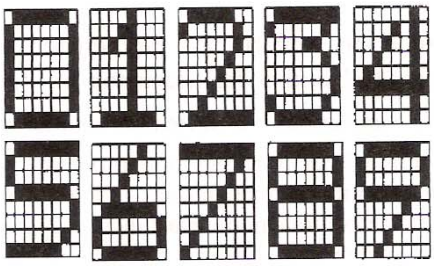


Fig. 13. Digits 0 to 9

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

6.3. Printed Persian digit recognition Problem: The ten printed Farsi digits are shown in figure 14. There is a page of 170 printed Farsi digits, 17 samples for every digit. 160 samples are used to train the network and the remaining samples are used for testing purpose. This page is digitized with resolution of 300 dpi. The momentum constants M_1 through M_7 as given below:

$$M_1 = \mu_{20} + \mu_{02}$$

$$M_2 = (\mu_{20} - \mu_{02})^2 + 4(\mu_{11})^2$$

$$M_3 = (\mu_{30} - 3\mu_{12})^2 + (\mu_{21} - 3\mu_{03})^2$$

$$M_4 = (\mu_{30} + 3\mu_{12})^2 + (\mu_{21} + 3\mu_{03})^2$$

$$M_5 = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})\left[(\mu_{30} + \mu_{12})^2 - 3(\mu_{12} + \mu_{03})^2\right] + (3\mu_{12} - 3\mu_{03})(\mu_{21} + \mu_{03})\left[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\right]$$

$$M_6 = (\mu_{30} - 3\mu_{02})\left[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\right] + 4\mu_{11}(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03})$$

$$M_7 = (3\mu_{21} - \mu_{03})(\mu_{30} + 3\mu_{12})\left[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2\right] - (\mu_{30} - 3\mu_{12})(\mu_{21} + \mu_{03})\left[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2\right]$$

Where μ_{pq} is scaled momentum from order p+q for a digit, are extracted from digitized images and submitted as inputs to the 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.

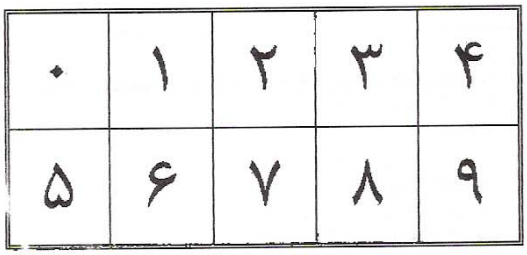


Fig. 14. Printed Persian digits

6.4. 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 [47]. First, voice signals are sampled with 44.1 kHz, then down sampled to 16 kHz. 14th –order LPC cepestral coefficients plus derivative LPC Cepestral Coefficients and delta log energy are extracted as speech features from each frame with length of 20 ms and shift of 10 ms. The dimension of feature vector extracted from each frame is 25. the speech samples are preemphasized with the filter coefficient 0.97 and Hamming windowed before calculating LPC cepestral coefficients. 938 training patterns are used in training phase. A three-layer TDNN[1] neural network with the delay of 3 frame is used.

In the following sections, simulation results for above problems for survival and also adaptive survival algorithms are given. In all the simulations, nonlinear sigmoid function is used as output and hidden layer activation functions, learning rate is 0.01 and number of iteration of BP algorithm in each step is 50.

6.5. Simulation results for survival algorithm: Weight determining algorithm for each application is implemented for 6 different networks with different number of hidden layer neurons. Simulation results are shown in tables 2 to 5.

6.5.1. English digit recognition problem: Off and on width coefficient are 10, weight reduction coefficient is 0.9 and momentum coefficient is 0.

Training patterns are fed 5000 times to network. Simulation results are in table 2.

6.5.2. Second order discrete time nonlinear function approximation problem: Off and on width coefficients are 10, weight reduction coefficient is 0.9 and momentum coefficient is

0.98. Training patterns are fed to network 500 times. Simulation results are shown in table 3.

6.5.3. Persian digit recognition problem: Off and on width coefficient are both 5, reduction coefficient is 0.9 and momentum coefficient is 0.98 Simulation results are shown in table 4.

6.5.4. Phoneme recognition Problem: Off and on width coefficient are 2 and 50 respectively, reduction coefficient 0.9 and momentum coefficient is 0.98. Simulation results are shown in table 5.

6.6. Simulation results for adaptive survival algorithm: for each problem, adaptive survival algorithm is implemented for 6 different networks with different number of hidden layer neurons. For all problems Tsetline automata is used for the purpose of adaptation of off and on width coefficient. Simulation results are shown in tables 2 through 5 and figures 15 and 20.

6.6.1. English digit recognition problem: {1, 2, 5, 10} is on width coefficient set and {10, 15, 20, 25} is off width coefficient set. Weight decrement coefficient and momentum coefficient is considered 0.9 and 0, respectively. Training patterns are fed 5000 times to the network. Simulation results are shown in table 2 and figures 15 and 16 show changes of on and off width coefficient with respect to epochs.

6.6.2. Second order discrete time nonlinear function approximation problem: In adaptive algorithm the maximum error belongs to network number 3 which is 2.2082. For this case, when sinusoidal input is used, waveforms of function output and network output are shown in figures 17 and 18 for two different frequencies. As shown in these figures, output of neural network tracks output of function as well. {4, 6, 8, 10} is on width coefficient set and {10, 12, 14, 16} is off width coefficient set. Weight decrement coefficient and momentum coefficients are chosen to be 0.9 and 0.98 respectively. Traning patterns are fed 5000 times to network. Simulation results are shown in table 3.

6.6.3. Persian Printed number recognition problem: {3.5, 4, 4.5, 5} is on width coefficient set and {15, 16, 17, 18} is off width coefficient set. Weight decrement coefficient and momentum coefficients are set to be 0.9 and 0.98, respectively. Simulation results are shown in table 4.

6.6.4. Phoneme recognition problem: {1, 25, 10} is on width coefficient set and {10, 15, 20, 25} is off width coefficient set. Weight decrement coefficient and momentum coefficients are set to be 0.9 and 0.98 respectively. Simulation results are shown in table 5.

For this problem another simulation is conducted and the changes in the number of weights as the algorithms proceeds is plotted (see figures 19, 20). We noted that in adaptive version a sharp decrease in the number of weights occurs in the beginning of the algorithm and then reduces slowly. In contrast to adaptive version of the algorithm, in non-adaptive version the number of weights reduces slowly at the beginning of the algorithm, but after some epochs it begins to go up, and remains up for a long period of time.

From above observation we may say that the adaptive version of the survival algorithm is a good candidate for parallel implementation [48].

Table 2. Comparing none adaptive with adaptive algorithm for English number recognition problem

Recognition Percentage (Test)	Recognition Percentage (learning)	Hidden layer Weights (adaptive algorithm)	Recognition Percentage (Test)	Recognition Percentage (learning)	Hidden layer Weights (none adaptive algorithm)	Hidden unit	Network
%100	%100	206	%100	%100	499	15	1
%90	%100	187	%90	%100	534	15	2
%90	%100	177	%100	%100	558	17	3
%100	%100	192	%100	%100	736	19	4
%100	%100	173	%100	%100	714	21	5
%90	%100	261	%90	%100	772	23	6
%95	%100	199.3	%96.7	%100	619	18.3	Ave.

Table 3. Comparing none adaptive with adaptive algorithm for Second order Discrete time nonlinear function approximation problem

NEURAL NETWORK ERROR FOR SINUSOIDAL INPUT	HIDDEN LAYER WEIGHTS (ADAPTIVE ALGORITHM)	NEURAL NETWORK ERROR FOR SINUSOIDAL INPUT	HIDDEN LAYER WEIGHTS (ADAPTIVE ALGORITHM)	HIDDEN UNIT	NETWORK
1.1162	19	0.3822	23	8	1
0.9409	22	0.9717	25	10	2
2.2082	28	2.7005	32	13	3
0.7325	38	0.6526	41	17	4
0.8012	43	0.6631	47	20	5
1.4580	57	1.3047	63	25	6
1.2095	34.5	1.1125	38.5	15.5	Ave.

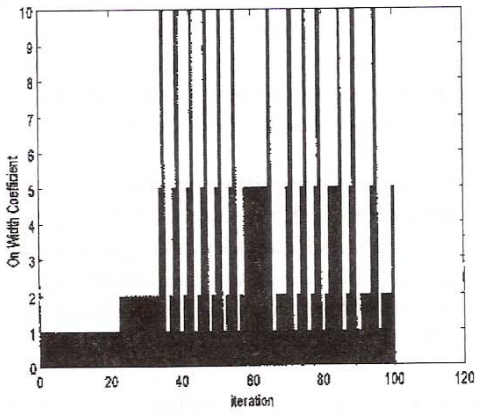


Fig. 15. On width coefficient changes

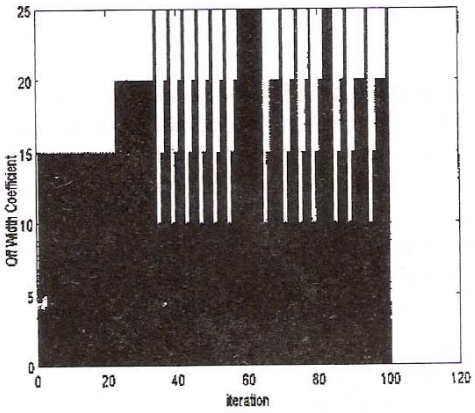


Fig. 16. Off width coefficient changes

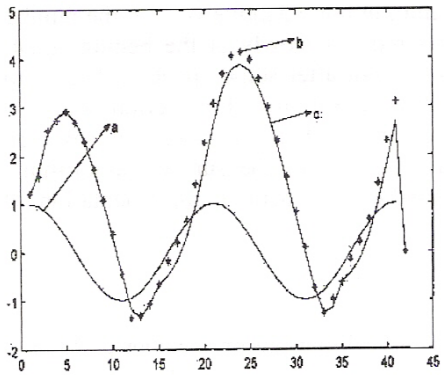


Fig. 17. a) Input b) function output c) network output

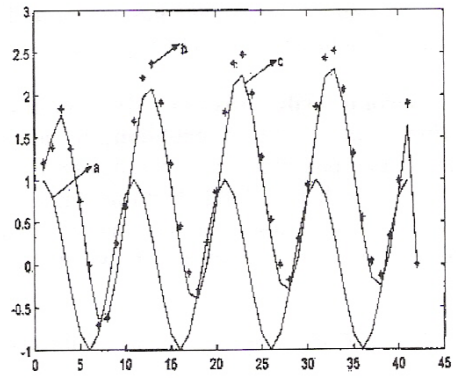


Fig. 18. a) Input b) function output c) network output

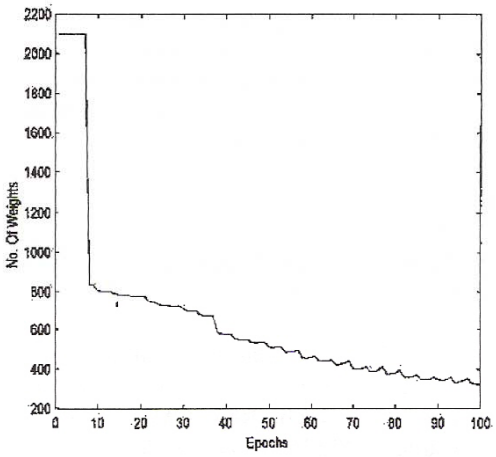


Fig. 19. Number of weights for Adaptive algorithm

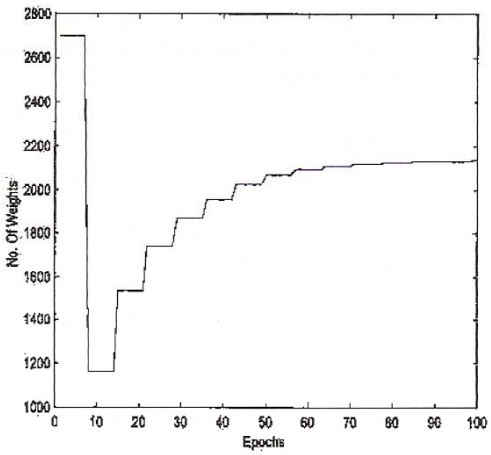


Fig. 20. Number of weights for none adaptive algorithm

Table 4. Comparing none adaptive with adaptive algorithm for Persian digit recognition problem

Recognition Percentage (Test)	Recognition Percentage (learning)	Hidden layer Weights (adaptive algorithm)	Recognition Percentage (Test)	Recognition Percentage (learning)	Hidden layer Weights (none adaptive algorithm)	Hidden unit	Network
%100	%100	11	%100	%100	26	6	1
%90	%100	19	%100	%100	23	7	2
%100	%100	29	%90	%100	47	12	3
%100	%100	29	%100	%100	32	12	4
%97.5	%100	22	%97.5	%100	32	9.25	Average

Table 5. Comparing none adaptive with adaptive algorithm for phoneme recognition problem

Recognition Percentage (Test)	Recognition Percentage (learning)	Hidden layer Weights (adaptive algorithm)	Recognition Percentage (Test)	Recognition Percentage (learning)	Hidden layer Weights (none adaptive algorithm)	Hidden unit	Network
%90	%100	191	%90	%100	945	16	1
%90	%100	238	%90	%100	1172	20	2
%90	%100	281	%85	%100	1418	24	3
%95	%100	353	%90	%100	1644	28	4
%90	%100	399	%95	%100	1855	32	5
%95	%100	470	%90	%100	2107	36	6
%92	%100	322	%90	%100	1523.5	26	Ave.

7. CONCLUSION

In this paper, we proposed an algorithm for adaptation of the parameters of the survival algorithm. The proposed algorithm tested on number of problems. It is shown through simulations that the network generated by the proposed algorithm has lesser number of weights and neurons, comparing to the first version of the survival algorithm reported earlier. Simulation results also show that the proposed algorithm has the same degree of generalization as the first version.

REFERENCES

[1] Tin-Yau Kwok and Dit-Yan Yeung, "Constructive Algorithms for structure Learning in Feed forward Neural Networks for Regression Problems", IEEE Trans. Neural Networks, Vol. 8, No. 3, May 1997, pp. 630-645.

[2] Meybodi, M. R. and Beigy. H., "Optimization of Neural Networks using Learning automata", Proc. Of 4th Annual Int. Computer Society of Iran Computer Conf. CSICC-98, Tehran, Iran, 1999, pp. 417-428, Iran (In Persian)

[3] Mozer, M. C. and Smolensky, P., "Skeletonization : A Technique for Trimming the Fat from a network via Relevance Assessment", in Advance in Neural Information Processing (1), D. S. Touretzhy, Ed. (Denveer 1988), 1989, pp. 107-115.

[4] Segee, B. E. and Carter, M. J., "Fault Tolerance of Pruned Multilayer Networks", in Proc. Int. Joint Conf. Neural Networks, Vol. II (Seattle), 1991, pp. 447-452.

[5] Karnin, E. D., "A Simple Procedure for Pruning Backpropagation Trained Neural Networks", IEEE Trans. Neural Networks, Vol. 1, No. 2, 1990, pp. 239-242.

[6] Le Cun, Y., Denker, J. S., and Solla, S. A., "Optimal Brain Damage", in Advances in Neural information Processing (2), D. S. Touretzky, Ed. (Denver 1989), 1990, pp. 598-605.

[7] Weigend, A. S., Rumelhart, D. E., and Huberman, B. A., "Generalization by Weight-Elimination with Application to Forecasting", in Advances in Neural Information Processing (3), R. Lippmann, J. Moody, and D. Touretzky, Eds., 1991, pp. 875-882.

[8] Weigend, A. S., Rumelhart, D. E., and Huberman, B. A., "Backpropagation, Weight-Elimination and Time Series Prediction", in Proc. 990 Connectionist Models Summer School, D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, Eds., 1990, pp. 105-116.

[9] Chauvin, Y., "A Backpropagation Algorithm with Optimal Use of Hidden Units", in Advances in Neural Information (1), D. S. Touretzky, Ed. (Denver 1988), 1989, pp. 519-526.

[10] Hanson, S. J., and Pratt, L. Y., "Comparing Biases for Minimal network Construction with Backpropagation", in Advances in Neural Information Processing (1), D. S. Touretzky, Ed (Denver 1988), 1989, pp. 177-185.

[11] Chauvin, Y., "Dynamic Behavior of Constrained Backpropagation Networks", in Advances in Neural Information Processing (2), D. S. Touretzky, Ed. (Denver 1989), 1990, pp. 642-649.

[12] C. Ji, Snapp, R. R., and Psaltis, D., "Generalizing Smothness Constrains from Discrete Samples", Neural Computation, Vol. 2, No. 2, 1990, pp. 188-197.

[13] Plaut, D. C., Nowlan, S. J., and Hinton, G. E., "Experiments on Learning by Backpropagation", Tech. Rep. CMU-CS-86-126, Carnegie-Mellon Univ., 1986.

[14] Nowlan, S. J., and Hinton, G. E., "Simplifying Neural Networks by soft Weight-Sharing", Neural Computation, Vol. 4, 1992, pp. 473-493.

[15] Marco Muselli, "On Sequential Construction of Binary Neural Networks", IEEE Trans. Neural Networks, Vol. 6, No. 3, May 1995, pp. 678-690.

[16] Steven Young and Tom Downs, "CARVE-A Constructive Algorithms for Real-Valued Examples", IEEE Trans. Neural Network, Vol. 9, No. 6, Nov. 1998, pp. 1180-1190.

[17] John, O., Moody and Panos, J. Antsakis, "The Dependence Identification Neural Network construction Algorithm", IEEE Trans. Neural Networks, Vol. 7, No. 1, Jan. 1996, pp. 3-15.

[18] Frattale Mascioli, F. M., and Martineli, G., "A Constructive Algorithm for Binary Neural Networks: The Oil-Sport Algorithm", IEEE Trans. Neural Networks, Vol. 6, No. 3, May 1995, pp. 794-797.

[19] Rudy Setiono and Lucas Chi Kwong Hui, "Use of a Quasi-Newton Method in a Feed forward Neural Network Construction Algorithm", IEEE Trans. Neural networks, Vol. 6, No. 1, Jan 1995, pp. 273-277.

[20] Nabhan T. M. and Zomaya A. Y., "Toward Neural Networks Structure for Function Approximation", Neural Networks, Vol. 7, No. 1, 1993, pp. 89-99.

[21] Hirose Y., Yamashita K., and Hijya S., "Backpropagation Algorithm which varies the Number of Hidden Units", Neural Networks, Vol. 4, No. 1, 1991, pp. 61-66.

[22] Schaffer, J. D., Whitely D., and Eshelman. L. J., "Combination of Genetic Algorithms and Neural Networks: A Survey of the State of the Art", IEEE Proc. COGANN-92, 1992, pp. 1-37.

[23] Maniezzo, V., "Genetic Evolution of the Topology and Weight Distribution of Neural Networks", IEEE Trans. On Neural Networks, Vol. 5, No. 1, 1994, pp. 39-53.

[24] Angeline, P. J., Saunders G. M., and Pollack, J. B., "Evolutionary Algorithms that Construct Recurrent Neural Networks", IEEE Trans. On Neural Networks, Vol. 5, No. 1, 1994, pp. 54-65.

[25] Yao X. and Liu Y., "A New Evolutionary System for Evolving Artificial Neural Networks", IEEE Trans. On Neural Networks, Vol. 8, No. 3, 1997, pp. 694-713.

[26] Whitley D., and Bogart C., "The Evolution of Connectivity" Pruning Neural Networks Using Genetic algorithms", Proc. Of Int. Joint Conf. On Neural Networks, Vol. 1, 1990, pp. 134.

[27] Beigy H. and Meybodi, M. R., "Optimization of Topology of Neural Networks Using Learning Automata", Proc. Of 4th Annual Int. Computer Society of Iran Computer Conf. CSICC-98, Tehran, Iran, 1999, pp. 417-428.

[28] Menhaj M. B., "Computational Intelligenece (Vol. 1) Fundamentals of Neural Networks", Professor Hessabi Publication, IRAN, Tehran, 1998.

[29] Chauvin, Y., "Generalization, Performance of Overtrained Backpropagation Networks", in Neural Networks, Proc. EURO-SIP Workshop, L. B. Al-Media and C. J. Wellekens, Eds., Feb. 1990, pp. 46-55.

[30] Levin E., Tishby N., and Solla S. A., "A Statistical Approach to Learning and Generalization in Layered Neural Networks", Proc. IEEE, Vol. 78, No. 10, oct. 1990, pp. 1568-1574.

[31] Schwartz, D. B., Samalan, V. K., Solla, S. A., and Denker, J. S., "Exhaustive learning" Neural Computation, Vol. 2, No. 1, 1990, pp. 374-385.

[32] Tishby, N., Levin, E., Solla, S. A., "Consistent Inference of Probabilities in Layered Networks: Predictions and Generalization", in Proc. Int. Joint Conf. Neural Networks, 1989, p. 403.

[33] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M., "Learnability and the Vapnik-Chervo Nenkis Dimention", J. Ass. Comput. Mach., Vol. 36, No. 4, 1989, pp. 929-965.

[34] Ehrenfeucht, A., Haussleer, D., Kearns, M., and Valiant, L., "A General Lower Bound on the Number of Examples Needed for Learning", In proc. 1988 Workshop computational Learning theory, 1988.

[35] Valiant, L. G., "A Theory of the Learnable", Commun.1 Ass. Comut. Mach., Vol. 27, No. 11, 1984, pp. 1134-1142.

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

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

[38] Meybodi, M. R., and Lakshmivarahan, S., "On a Class of Learning Algorithm which have a Symmetric Behavior under Success and Failure", Springer Varleg Lecture Notes in Statistics, 1984, pp. 145-155.

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