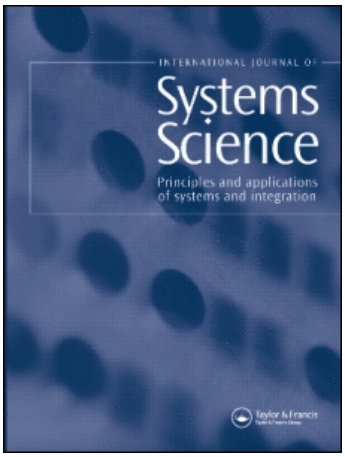


International Journal of Systems Science
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713697751

A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks

Hamid Beigy [ab]; Mohamad Reza Meybodi [bc]
[a] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran [b] School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics (IPM), Tehran, Iran [c] Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran

First Published:January2009

To cite this Article Beigy, Hamid and Meybodi, Mohamad Reza(2009)'A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks',International Journal of Systems Science,40:1,101 — 118
To link to this Article: DOI: 10.1080/00207720802145924
URL: http://dx.doi.org/10.1080/00207720802145924

International Journal of Systems Science
Vol. 40, No. 1, January 2009, 101–118

Taylor & Francis
Taylor & Francis Group

A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks

Hamid Beigy[ab]* and Mohamad Reza Meybodi[bc]

[a]*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran;* [b]*School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics (IPM), Tehran, Iran;* [c]*Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran*

(*Received 5 November 2003; final version received 16 April 2008*)

There is no method to determine the optimal topology for multi-layer neural networks for a given problem. Usually the designer selects a topology for the network and then trains it. Since determination of the optimal topology of neural networks belongs to class of NP-hard problems, most of the existing algorithms for determination of the topology are approximate. These algorithms could be classified into four main groups: pruning algorithms, constructive algorithms, hybrid algorithms and evolutionary algorithms. These algorithms can produce near optimal solutions. Most of these algorithms use hill-climbing method and may be stuck at local minima. In this article, we first introduce a learning automaton and study its behaviour and then present an algorithm based on the proposed learning automaton, called survival algorithm, for determination of the number of hidden units of three layers neural networks. The survival algorithm uses learning automata as a global search method to increase the probability of obtaining the optimal topology. The algorithm considers the problem of optimization of the topology of neural networks as object partitioning rather than searching or parameter optimization as in existing algorithms. In survival algorithm, the training begins with a large network, and then by adding and deleting hidden units, a near optimal topology will be obtained. The algorithm has been tested on a number of problems and shown through simulations that networks generated are near optimal.

Keywords: neural networks engineering; multi-layer 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, to mention a few. Among them, the class of multi-layer feed forward networks, which use backpropagation algorithm for training, is perhaps the most popular. Methods using standard backpropagation algorithm perform gradient descent only in the weight space of a network with fixed topology. 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 over fitting and poor generalization performance.

Since the problem of determination of optimal topology for neural networks belongs to the class of NP-hard problems, most existing algorithms for determination of the topology are approximate. These algorithms could be classified into four main groups: pruning algorithms, constructive algorithms, hybrid algorithms and evolutionary algorithms. These algorithms can produce near optimal solutions. In the next few paragraphs we briefly explain these algorithms.

Pruning algorithms: Pruning algorithms start with a large 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 small networks (higher generalization ability) (Kruschke 1988, 1989; Reed 1993; Castellano, Fanelli, and Pelillo 1997).

Constructive algorithms: These algorithms start with a small initial network and gradually add new hidden units or layers until learning take places (Mezard and Nadal 1989; Frean 1990; Fahlman and Lebier 1990; Marchand, Golea, and Rujan 1990; Sirat and Nadal 1990; Yeung 1991; Meltser, Shoham, and Manevitz, (1996); Kwok and Yeung 1997; Beigy and Meybodi 1998).

*Corresponding author. Email: beigy@sharif.edu

ISSN 0020–7721 print/ISSN 1464–5319 online

DOI: 10.1080/00207720802145924
http://www.informaworld.com

Hybrid algorithms: Hybrid algorithms are a combination of pruning and constructive algorithms. These algorithms try to attain a satisfactory solution by adding or removing hidden units and weights (Hirose, Yamashita, and Hijya 1991; Nabhan and Zomaya 1994).

Evolutionary algorithms: These algorithms use a performance index such as minimum error, and use evolutionary algorithms to search in parameter space for optimal structure. In these algorithms, each point of the search space corresponds to a network structure (Whitley and Bogart 1990; Schaffer, Whitely, and Eshelman 1992; Angeline, Saunders, and Pollack 1994; Maniezzo 1994).

Most of these algorithms use hill-climbing method and may be stuck at local minima. In this article, we first introduce a learning automaton and study its behaviour and then propose an algorithm based on learning automata for determination of the number of hidden units of three layers neural networks. We call this algorithm survival algorithm. The proposed algorithm uses the proposed learning automaton as a global search tool to increase the probability of obtaining the optimal topology. In the survival algorithm, the training begins with a large network, and then by adding and deleting hidden units, the optimal topology is obtained. The proposed algorithm has been tested on a number of problems and shown through simulations that networks generated are near optimal.

The rest of the article is organised as follows: the next section briefly presents the standard backpropagation algorithm and learning automata. The proposed learning automaton and analysis of its behaviour in stationary environments are given in 'An OMA for determining the number of units in hidden layer' and 'Behavior of HULA in a stationary environment'. The proposed algorithm for finding the optimal topology is given in 'The survival algorithm'. Simulation results are presented in 'Simulation results' and 'Conclusion' concludes the article.

2. Backpropagation algorithm and learning automata

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

2.1. *Backpropagation algorithm*

Error backpropagation training (BP) algorithm, which is an iterative gradient descent algorithm, is a simple way to train multi-layer feed forward neural networks (Rumelhart, Hinton, and Williams 1986). The BP algorithm is based on the following gradient descent rule:

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

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

$$G(n) = -\nabla E_p(n). \quad (2)$$

E_p is the sum of squared error given by:

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

where $T_{p,j}$ and $O_{p,j}$ are desired and actual outputs for pattern p at output node j, respectively. The efficiency of the BP algorithm for a particular application is largely dependent on the topology (number of layers, number of neurons in each layer and connections between layers) of the network. In order to evaluate the topology of a neural network with specific learning algorithm, several criteria have been suggested. In what follows, we describe some of them.

***Training complexity*:** Training complexity is defined as the training time for a given training data. Although the training time is problem dependent, the determination of the optimal values for weights for a given training data belongs to a class of NP-complete problems (Judd 1990). Hence approximate algorithms such as error back propagation, which is based on gradient descent, are proposed for the determination of the appropriate values of weights. For these algorithms the exact running time depends on the shape of error surface, which depends on the topology of the network. Unfortunately, for some special networks, the shape of error surface can be determined (Yu 1992).

***Generalization ability*:** Generalization ability of the network is estimated based on its performance on previously unseen training patterns. The generalization ability of a network for a given problem depends on the following four factors: size and goodness of training data, architecture and topology of the network, training algorithm and complexity of the given problem. For the above four factors, the designer of the network has no control on the complexity of the problem. Hence after fixing the training algorithm, the generalization ability of a neural network can be studied from the following points of view.

Fix training data and use different network topology to obtain a network with appropriate performance. For example constructive and pruning algorithms use such an approach. We also use this approach with a different idea.

Fix (use different) network topology and use different training data to obtain a network with higher generalization ability. For example *cross validation* uses such an approach.

***Memorization capacity*:** Memorization capacity is the ability of the network performance on training patterns. Memorization capacity is indicative of a network error for training data. If the error is high then memorization capacity is low and if the error is low, memorization capacity is high, that is, the memorization capacity is inversely proportional to the network error.

2.2. *Learning automata*

Learning of an automaton involves determination of an optimal action from a finite set of actions. Automaton selects an action from its finite set of actions and applies it to a random environment, which in turn emits a stochastic response $\beta(n)$ at the time n. $\beta(n)$ is an element of $\underline{\beta}=\{0,1\}$ and is the feedback response of the environment to the automaton. The environment penalises (i.e., $\beta(n)=1$) action α_i of the automaton with probability c_i. On the basis of the response $\beta(n)$, the state of automaton is updated and a new action chosen at the time $(n+1)$. Note that the $\{c_i\}$ are unknown initially and it is desired that as a result of interaction with the environment the automaton arrive at the action, which presents it with the minimum penalty in an expected sense. With no *a priori* information, the automaton chooses its actions with equal probability. Initially, the expected penalty is M_0, which is the mean of penalty probabilities. We denote the expected penalty at time n as $E[M(n)]$. An automaton is said to learn expediently if $\lim_{n\to\infty} E[M(n)] < M_0$.

The automaton is optimal if $E[M(n)]$ equals the minimum of $\{c_i\}$ as time tends to infinity. It is ε-optimal if for any arbitrary $\varepsilon>0$, $E[M(n)]<c_1+\varepsilon$ as time goes toward infinity, where $c_1=\min_i\{c_i\}$. The ε-optimality could be achieved by a suitable choice of some parameter of the automaton.

Learning automata (LA) can be classified into two main families, fixed and variable structure automata (Mars, Narendra, and Chrystall 1983; Oommen and Ma 1988; Oommen, Valiveti, and Zgierski 1991; Oommen and Croix 1996). Examples of the fixed structure type are Tsetline, Krinsky, TsetlineG and Krylov automata and examples of variable structure type are linear reward-inaction and linear reward-penalty algorithm (Narendra and Thathachar 1989).

Object migrating automata: Object migrating automaton (OMA), which is a type of automata, can be shown by a sextuple$\langle \underline{\alpha}, \underline{W}, \underline{\Phi}, \underline{\beta}, F, G$, where: $\alpha=\{\alpha_1,\ldots,\alpha_r\}$ is the set of actions, $\underline{W}$ is the set of objects, $\underline{\Phi}=\{\phi_1,\ldots,\phi_s\}$ is the set of internal states, $\underline{\beta}=\{0,1\}$ is the set of inputs, $F:\underline{\Phi}\times\underline{\beta}\to\underline{\Phi}$ is the state transition map, and $G:\underline{\Phi}\to\underline{\alpha}$ is the output map (Oommen and Ma 1988; Oommen et al. 1991; Oommen and Croix 1996). This automaton is used for object partitioning (Oommen and Ma 1988), keyboard optimization (Oommen et al. 1991), graph partitioning (Oommenand Croix 1996) and graph isomorphism (Beigy and Meybodi 2000), to mention a few. In this automaton, each action shows one class. In automaton with fixed structure, response of environment to the automaton changes state of automaton. While in object migration automaton, objects are associated to states of the automaton and response of environment to automaton causes migration of object between states of automaton. This migration does object classification. If object W_i lies in action α_j, then it belongs to class j. For action α_k state set are considered $\{\phi_{(k-1)N+1},\ldots,\phi_{kN}\}$, 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 two objects W_i and W_j are in states $\phi_{(k-1)N+1}$ and $\phi_{(k-1)N+m}$ respectively, then the membership probability of W_i is larger than that of W_j. Therefore, for action α_k, state $\phi_{(k-1)N+1}$ is called the state with highest probability of membership and state ϕ_{kN} is called the state with lowest probability of membership.

Learning automata have been used in many applications such as: data compression (Hashim, Amir, and Mars 1986), queuing theory (Meybodi and Lakshmivarhan 1983), telephone traffic control (Narendra and Thathachar 1989), solving NP problems (Oommen and Ma 1988; Oommen et al. 1991; Oommen and Croix 1996), pattern recognition (Thathachar and Sastry 1987), control of computer networks (Mars et al. 1983; Mars, Chen, and Nambiar 1998), adaptation of parameters of neural networks (Beigy and Meybodi 2000, 2001a,b Meybodi and Beigy 2002a,b; Beigy, Meybodi, and Menhaj 2002; Adibi, Meybodi, and Safabaksh 2005), and call admission in cellular networks (Beigy and Meybodi 2002a,b, 2003, 2004, 2005) to mention a few. For more information about learning

automata refer to Narendra and Thathachar (1989; and Mars et al. (1998).

3. An OMA for determining the number of units in hidden layer

In this section, we introduce an object migrating automaton called hidden units learning automaton (HULA) for determination of near optimal topology (i.e. a topology with minimum number of hidden units) in three layer neural networks. HULA can be represented by a sextuple $\langle \underline{\alpha}, \underline{H}, \underline{\Phi}, \underline{\beta}, F, G \rangle$ where

(1) $\underline{\alpha} = \{\alpha_1, \alpha_2\}$ is the set of allowable actions. This automaton has two actions: the first action is for **on** hidden units. Units associated to this action are used for training the neural network. The second action is for **off** hidden units. Units associated to this action do not participate in training the neural network.

(2) $H = \{H_1, H_2, \ldots, H_M\}$ is the set of hidden units which are associated to the states of HULA. If unit H_i appears on states of action α_1, then this unit is considered an **on** unit and if it appears on states of action α_2 it is considered an **off** unit.

(3) $\underline{\Phi} = \{\phi_1, \ldots, \phi_{2N}\}$ is the set of states of automaton. The set of states is partitioned into two subsets $\{\phi_1, \ldots, \phi_N\}$, $\{\phi_{N+1}, \ldots, \phi_{2N}\}$, where N is the memory depth of automaton. The set of on units and the set of off units are defined as $ON = \{H_i | 1 \leq \text{State } (H_i) \leq N\}$ and $\text{OFF} = \{H_i | N+1 \leq \text{State } (H_i) \leq 2N\}$, where State$(H_i)$ is the state on which H_i is placed. If H_i belongs to set of states $\{\phi_1, \ldots, \phi_N\}$ then this unit is considered as an on unit. If a unit is placed on state ϕ_1, then it is considered as a unit with the highest degree of importance. If it is placed on state ϕ_N, then it is considered as a unit with the lowest degree of importance. If unit H_i belongs to set $\{\phi_{N+1}, \ldots, \phi_{2N}\}$ the unit is considered as an off unit. If a unit is placed on state ϕ_{N+1}, the unit has the highest degree of importance and if it is placed on state ϕ_{2N}, it has the lowest degree of importance.

(4) $\underline{\beta} = \{0, 1\}$ is the set of input to HULA; **1** represents failure and **0** represents success.

(5) $F: \underline{\Phi} \times \underline{\beta} \to \underline{\Phi}$ is the state transition function. This function shows how hidden units migrate among the states of automaton to produce different partitioning. Description of this function will be given later in this article.

(6) $G: \underline{\Phi} \to \underline{\alpha}$ is the output function. This function determines what state is associated to which action. If state of a unit belongs to set $\{\phi_1, \ldots, \Phi_N\}$, then that unit is associated to action α_1; otherwise it is associated to action α_2.

For simplicity in presentation, a HULA with K actions, memory depth of N, and M hidden units, will be denoted by HULA (K, N, M). To describe the state function of HULA, we consider four different cases as described below.

If H_i is an **on** unit and placed on state ϕ_j (for $i = 1, 2, \ldots, N$) and receives a reward, then its degree of importance will be increased, that is, it will move toward internal states of action α_1. Such a movement is shown in Figure 1. If hidden unit H_i is placed on state ϕ_1 and receives a reward then its state does not change.

If H_i is an **on** unit, placed on state ϕ_j (for $i = 1, 2, N$) and gets a penalty, then its degree of importance will be reduced, that is, it moves toward boundary states of action α_1. Examples of such a movement are depicted in Figures 2 and 3.

If H_i is an **off** unit and is placed on state ϕ_j and gets a penalty, then it moves toward **on** units, that is, it moves towards boundary states. Examples of such movements for two different cases are depicted in Figures 4 and 5.

If H_i is an **off** unit and placed on state ϕ_j and receives a reward, then its degree of importance is increased and moves away from **on** units, that is, it moves towards internal states of automaton. An example of such movement is shown in Figure 6. If H_i

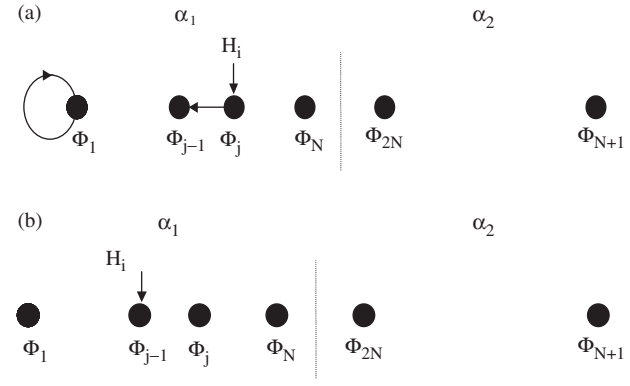


Figure 1. How to reward an **on** unit: (a) before reward and (b) after reward.

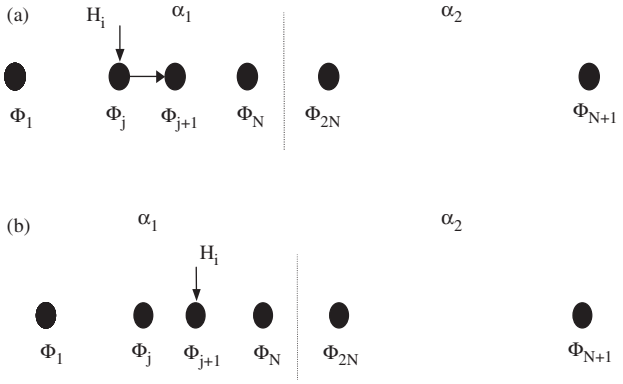


Figure 2. How to penalise an **on** unit for a non-boundary state: (a) before penalty and (b) after penalty.

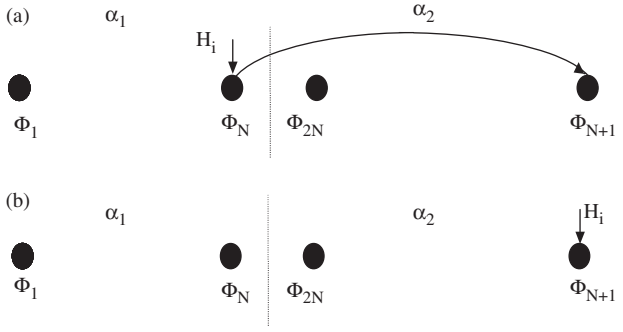


Figure 3. How to penalise an **on** unit for a boundary state: (a) before penalty and (b) after penalty.

is placed on state ϕ_{N+1} and receives a reward then it remains in that state.

4. Behaviour of HULA in a stationary environment

In this section we study the steady state behaviour of HULA (2, N, 1) in a static environment. Transition function for HULA (2, N, 1) is given in Figure 7. The action taken by automaton HULA (2, N, 1) indicates whether its associated hidden unit is **on** or **off**. The behaviour of HULA (2, N, M) is equivalent to the behaviour of M automata HULA (2, N, 1) participating in a cooperative game of learning automata in order to determine the optimal topology. In this game, automaton A_i determines the status of hidden unit H_i. If automaton A_i is in state Φ_m then hidden unit H_i is associated to state Φ_m. If automaton A_i is in one of states $\Phi_1, \ldots, \Phi_N$, then H_i is **on** and if automaton A_i is in one of states $\Phi_{N+1}, \ldots, \Phi_{2N}$, then unit H_i is **off**.

The next theorem proves the ε-optimality of HULA (2, N, 1) in stationary random environments.

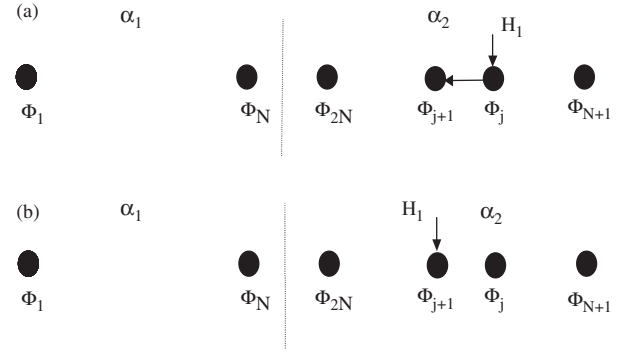


Figure 4. How to penalise an **off** unit for a non-boundary state: (a) before penalty and (b) after penalty.

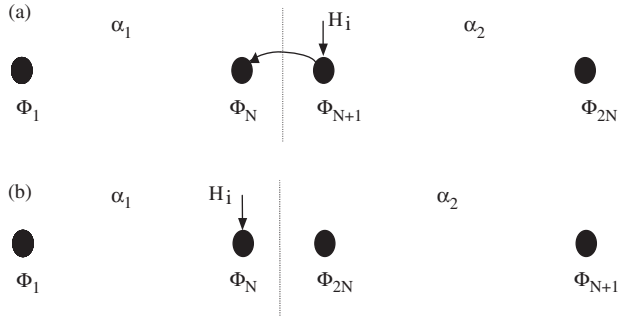


Figure 5. How to penalise an **off** unit for a boundary state: (a) before penalty and (b) after penalty.

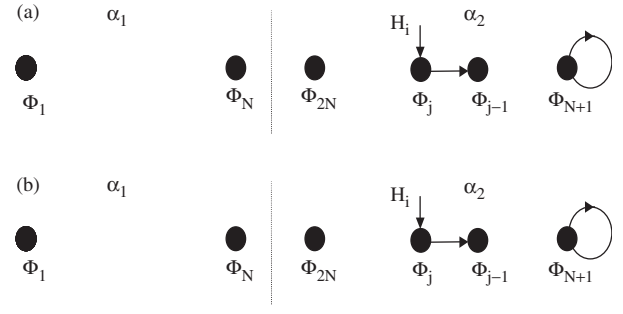


Figure 6. How to reward an **off** unit: (a) before reward and (b) after reward.

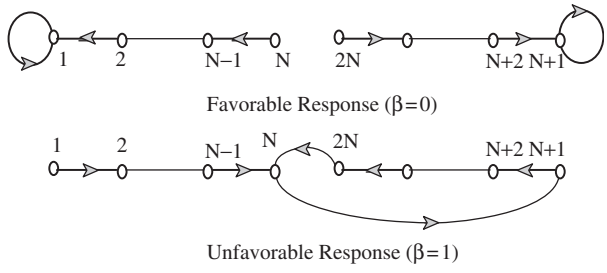


Figure 7. State transition graph for HULA (2, N, 1).

Theorem 1: If $\theta = c_1/c_2 < 1$ then, HULA (2, N, 1) is ε-optimal.

Proof: Using the transition function for HULA $(2, N, 1)$ described in the previous section we have

$$F(0) = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & \vdots & & \vdots & & & & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & & \vdots & \vdots & & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix}$$

$$F(1) = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & & \vdots & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & \vdots & & \cdots & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & & 0 & \vdots & & & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

Thus matrix of transition probability for HULA (2, N, 1) is

$$P = \begin{bmatrix} d_1 & c_1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ d_1 & 0 & c_1 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & d_1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & & & & & \vdots & \vdots & & & & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & c_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & d_1 & 0 & c_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & d_2 & c_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & d_2 & 0 & c_2 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & d_2 & 0 & \cdots & 0 & 0 \\ \vdots & & & & & \vdots & \vdots & & & & & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & d_2 & 0 & c_2 \\ 0 & 0 & 0 & \cdots & 0 & c_2 & 0 & \cdots & 0 & 0 & d_2 & 0 \end{bmatrix}$$

where $d_i = 1 - c_i$ is reward probability of action α_i; Behaviour of this automaton in static environment can be described by an ergodic Markov chain. The vector of steady state probabilities $\Pi = [\pi_1, \ldots, \pi_{2N}]^T$ of this Markov chain can be computed using equation $P^T\Pi = \Pi$. To simplify the analysis, we assume $c_2 = d_1$ and $c_1 = d_2$. Using equation $P^T\Pi = \Pi$ we have

$$\begin{aligned} c_1\pi_1 + c_2\pi_2 &= \pi_1 && (1) \\ &\vdots \\ c_1\pi_{k-1} + c_2\pi_{k+1} &= \pi_k && (k) \\ &\vdots \\ c_1\pi_{N-1} + c_2\pi_{2N} &= \pi_N && (N) \\ c_1\pi_N + c_1\pi_{N+1} + c_1\pi_{N+2} &= \pi_{N+1} && (N+1) \\ &\vdots \\ c_1\pi_{N+k-1} + c_2\pi_{N+k+1} &= \pi_{N+k} && (N+k) \\ &\vdots \\ c_2\pi_{2N-1} &= \pi_{2N}, && (2N) \end{aligned} \tag{4}$$

Equations (k) and $(N+k)$ are difference equations of second order and Equations (1), (N), $(N+1)$ and $(2N)$ give the boundary conditions. To solve the above system of equations we assume that the solution has the following form.

$$\pi_k = A_1\lambda_1^{k-1} \quad \text{and} \quad \pi_{N+k} = A_2\lambda_2^{k-1}$$

Substituting the above two equations in equations (k) and $(N+k)$ for $k > 1$, we obtain characteristic equation

$$(1 - c_m)\lambda_m^2 - \lambda_m + c_m = 0, \quad m = 1, 2.$$

Solving the above characteristic equations, we obtain

$$\begin{aligned} &\lambda_1^{(1)} = 1, \quad \lambda_1^{(2)} = \frac{c_1}{1 - c_1} = \theta, \\ &\lambda_2^{(1)} = 1 \quad \text{and} \quad \lambda_2^{(2)} = \frac{1 - c_1}{c_1} = \frac{1}{\theta}. \end{aligned} \tag{5}$$

Substituting the above values in (4), we obtain

$$\begin{aligned} \pi_k &= A_1\theta^{k-1} + B_1 \quad \text{and} \\ \pi_{N+k} &= A_2\theta^{k-1} + B_2 \quad \text{for} \quad k = 1, 2, \ldots, N. \end{aligned} \tag{6}$$

Using (4) and (6), we obtain $B_1 = 0$, $B_2 = A_2/\theta^N$, and $A_2 = A_1(\theta^{2N}/(\theta - 1))$. Therefore, we have

$$\begin{aligned} \pi_k &= A_1\theta^{k-1} && k = 1, 2, \ldots, N \\ \pi_{N+k} &= \frac{A_1}{\theta - 1}\theta^N[\theta^{N-k+1} - 1] && k = 1, 2, \ldots, N. \end{aligned}$$

Hence P_1 and P_2 can be computed as follows

$$P_1 = \sum_{k=1}^{N} \pi_k = A_1 \frac{\theta^N - 1}{\theta - 1}$$

$$P_2 = \sum_{k=1}^{N} \pi_{N+k} = \frac{A_2}{\theta^N}\left[\frac{\theta(1-\theta^N) - N(1-\theta)}{(1-\theta)}\right]$$

Using values of P_1 and P_2 and the fact that $P_1 + P_2 = 1$ we have

$$A_1 = \frac{(\theta-1)^2}{(\theta-1)(\theta^N-1) + N\theta^N(1-\theta) - \theta^{N+1}(1-\theta^N)}$$

$$A_2 = \frac{\theta^{2N}(\theta-1)}{(\theta-1)(\theta^N-1) + N\theta^N(1-\theta) - \theta^{N+1}(1-\theta^N)},$$

and hence

$$P_1 = \frac{(\theta^N-1)(\theta-1)}{(\theta-1)(\theta^N-1) + N\theta^N(1-\theta) - \theta^{N+1}(1-\theta^N)}$$

$$P_2 = \frac{\theta^N\left[N(1-\theta) - \theta(1-\theta^N)\right]}{(\theta-1)(\theta^N-1) + N\theta^N(1-\theta) - \theta^{N+1}(1-\theta^N)}.$$

Therefore, the average penalty for HULA $(2, N, 1)$ is

$$\begin{aligned} M_{\text{HULA }(2,N,1)} &= c_1P_1 + c_2P_2 = c_1P_1 + (1-c_1)P_2 \\ &= \frac{\theta}{1+\theta}P_1 + \frac{1}{1+\theta}P_2, \end{aligned}$$

or

$$M_{\text{HULA }(2,N,1)} = \frac{1}{1+\theta}\frac{\theta(\theta-1)(\theta^N-1) + N\theta^N(1-\theta) - \theta^{N+1}(1-\theta^N)}{(\theta-1)(\theta^N-1) + N\theta^N(1-\theta) - \theta^{N+1}(1-\theta^N)}.$$

If $\theta < 1$, then we have

$$\lim_{N\to\infty} M_{\text{HULA }(2,N,1)} = \frac{\theta}{1+\theta} = \min\{c_1, c_2\} = c_1,$$

and hence the ε-Optimality of HULA $(2, N, 1)$. □

5. The survival algorithm

In this section, we introduce an algorithm for determination of the number of hidden units in three layer neural networks. This algorithm uses a HULA presented before and backpropagation algorithm and finds a near optimal topology while the network is being trained. The proposed algorithm considers the problem of optimization of the topology of neural networks as object partitioning rather than searching or parameter optimization as in existing algorithms. The goal is to find the smallest set of hidden units, which is able to solve the given problem. The algorithm on the basis of the performance of each unit adaptively changes the number of hidden units as the network being trained to find a network with near optimal topology.

This algorithm works as follows: initially, all hidden units are on and placed on state ϕ_1. Using these hidden units, the network will be trained for a specific period of time. After this period is over, those hidden units which have not performed well will receive a penalty, those hidden units which have performed well will receive a reward, and those hidden units about which a definite judgment cannot be made neither penalised nor rewarded.

The variance of the activation of a unit, which we called energy used by the unit, is used as a measure of the performance for that unit. A unit has performed well if the energy is used by that unit is high, which means that information stored in weights of this unit are important. A unit has not performed well if the energy is used by that unit is low, which means that information stored in weights of this unit are not important. In what follows, we describe how to discriminate among the on or off units.

How to discriminate among on *units*: A unit whose used energy is less than a threshold value is considered as an inappropriate unit and a unit whose used energy is larger than another threshold is considered as an appropriate unit. To determine these two thresholds, the variance of activation of that unit is computed as follows:

$$\sigma_I = \sqrt{\frac{\sum_{K=1}^{P}(|U_{IK}| - \mu_I)^2}{P}} \quad I \in ON,$$

where U_{IK} is the activation value of on hidden unit H_I for training pattern K and P is the number of training patterns. μ_I is the average of activation of on hidden unit H_I defined as

$$\mu_I = \frac{\sum_{K=1}^{P}|U_{IK}|}{P} \quad I \in ON.$$

On units whose variance of their activation is less than a threshold value are penalised, on units whose variance of their activation is larger than another threshold value are rewarded, and on units whose variance of their activation are between these two thresholds neither rewarded nor penalised (Figure 8).

M_{ON} is the average of variances of on hidden units and computed by the following expression

$$M_{ON} = \frac{\sum_{K\in ON}\sigma_K}{|ON|},$$

where $|ON|$ is the number of elements of set ON. The width X_{ON} in Figure 8 is computed as follows

$$X_{ON} = \lambda_{ON} \frac{|ON| + |OFF|}{|ON|} \times \frac{\text{Max}(\sigma_{ON})}{\text{Min}(\sigma_{ON})}.$$

In the above equation, $\lambda_{ON} \geq 0$ is called the coefficient of on units' width. $\text{Max}(\sigma_{ON})$ and $\text{Min}(\sigma_{ON})$ are maximum and minimum variances of activation of on units, respectively. Units whose variance of their activation is less than $M_{ON} - X_{ON}$ are penalised, units whose variance of their activation is larger than $M_{ON} + X_{ON}$ are rewarded and on units whose variance of their activation lie in the range of $[M_{ON} - X_{ON}, M_{ON} + X_{ON}]$ are neither rewarded nor penalised.

How to discriminate among off *units*: Since *Off* units don't participate in training, their activation and variance of activation don't exist. For this reason, we use their past activation and variance of activation to determine the present state of that unit. Activation of an off unit for a pattern is computed based on the activation of that unit when it was on last time. If a unit has been off for a long period then its activation value will be reduced because of the fact that the point on which it is placed on the error curve is probably closer to the answer point than the point when it was on. Therefore, activation of an off unit can be computed as below:

$$U_{IK}(n+1) = U_{IK}(n) e^{-\lambda_d |U_{IK}(n)|}$$

In the above equation, constant $\lambda_d \geq 0$ is called the coefficient of activation reduction and n is the time index. According to the above equation the variance of activation of an off unit will be reduced gradually as the algorithm proceeds. The variance of off units is computed as follows:

$$\sigma_I = \sqrt{\frac{\sum_{K=1}^{P} (|U_{IK}| - \mu_I)^2}{P}} \quad I \in OFF,$$

where U_{IK} is the activation value of off unit H_I for pattern K. μ_I is the average of activation of off unit H_I, which is computed by the following expression.

$$\mu_I = \frac{\sum_{K=1}^{P} |U_{IK}|}{P} \quad I \in OFF.$$

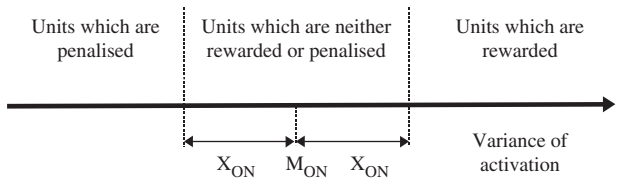


Figure 8. Classification of on units.

Once the variance is computed off units whose variance of activation is less than a threshold value are rewarded, off units whose variance of activation is larger than a another threshold value are penalised, and off units whose variance of their activation are between these two thresholds neither rewarded nor penalised (Figure 9).

M_{OFF} is the average of variances of off units and computed as follows:

$$M_{OFF} = \frac{\sum_{K \in OFF} \sigma_K}{|OFF|},$$

where $|OFF|$ is the number of elements of set OFF. The width X_{OFF} in the above figure is computed as follows

$$X_{OFF} = \lambda_{OFF} \frac{|ON| + |OFF|}{|OFF|} \times \frac{\text{Max}(\sigma_{OFF})}{\text{Min}(\sigma_{OFF})}.$$

In the above equation $\lambda_{OFF} \geq 0$ is called coefficient of off units' width. Max (σ_{OFF}) and Min (σ_{OFF}) are called maximum of variance of activation of off units and minimum of variance of activation of off units, respectively. Off units whose variance of activation is less than $M_{ON} - X_{ON}$ are rewarded. Those off units whose variance of activation is larger than $M_{ON} + X_{ON}$ are penalised, and those off units whose variance of activation lies in the range of $[M_{OFF} - X_{OFF}, M_{OFF} + X_{OFF}]$ are neither rewarded nor penalised.

In order to take advantage of large networks during the training, the sensitivity of all neurons is considered very small at the beginning of the training. This causes low competition among neurons and hence reduces the complexity of training, and increases the probability of escaping from local minima. When the network produces a reasonable output the sensitivity of neurons is increased gradually in order to achieve higher competition among neurons. To increase the sensitivity of the neurons we increase the steepness of the activation function. For the algorithm to regulate the steepness of the activation function we regulate the steepness of the activation function by the following equation.

$$\gamma = e^{-\lambda_s \times \text{MSE}},$$

where γ is the steepness of sigmoid function and $\lambda_s \geq 0$ is the coefficient of steepness variation and MSE is the mean square error. In order to compute the starting

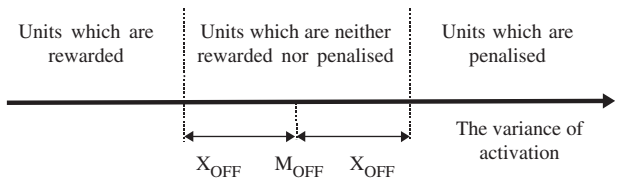
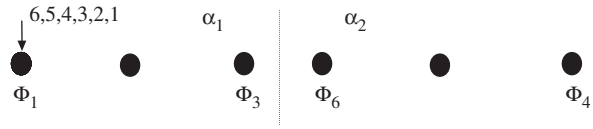


Figure 9. Classification of off units.

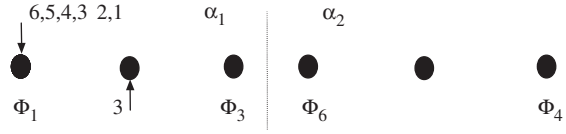
weights for those units whose mode is changed from off to on we use weights at the last epoch that the unit was on. The time complexity of an iteration of survival algorithm for a network with I_N input units, H_N hidden units, O_N output units, K epochs of training, and P training samples is equal to $\theta(KP(1+I_N)(1+H_N)O_N)$. Survival algorithm is given in Figure 10.

Example: To show how the proposed algorithm works we give an example. For this example we use HULA (2, 6, 6) with $\lambda_{on}=0.05$ and $\lambda_{off}=0.01$. The variance of activation of hidden units and how the automaton works for the first eight steps are given below (Table 1).

Step 1: At the beginning of the training all hidden units are placed on state ϕ_1. At the end of this step, states of the automaton and their associated hidden units are given below:



Step 2: In this step, unit 3 is penalised, unit 6 is rewarded, and remaining units do not change their states. At the end of this step, states of the automaton and their associated hidden units are given below:



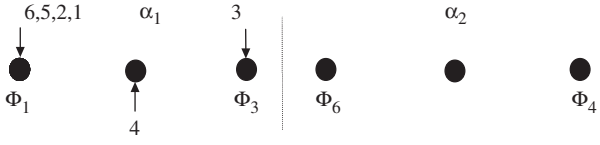

```
Algorithm Survival
  input:
    Training Set (X, T)
    Maximum No. of Hidden Units: Hmax
  output:
    Network Weight Vector: W
    Network Topology: Set of ON
  repeat
    for m := 1 To K do
      call BP // After K steps hidden units
are examined
    end for
    // Decrease the activation of off units
    for all I ∈ OFF do
      for k := 1 To P do
        Uik := exp(- λd | Uik| )
      end for
    end for
    for I := 1 To Hmax do
      Compute σI
    end for
    Comput MON, MOFF, XOFF, XON
    // Move the hidden units among automata's
states
    for I := 1 to Hmax do
      if I ∈ ON then
        if σI < (MON- XON) then
          call PenaliseOnUnit (I)
        end if
        if σI > (MON+ XON) then
          call RewardOnUnit (I)
        end If
      end If
      if I ∈ OFF then
        if σI < (MOFF- XOFF) then
          call RewardOFFUnit (I)
        end if
        if σI > (MOFF+ XOFF) then
          call PenaliseOFFUnit (I)
        end if
      end if
    end for
  until Termination Condition is Satisfied.
  return (W, ON)
end Algorithm

procedure PenaliseOnUnit (I)
  inc (State (I))
end procedure

procedure RewardOnUnit (I)
  if State (I) > 1 then
    dec (State (I))
  end if
end procedure

procedure PenaliseOffUnit (I)
  if State (I) ≠ 2 * N then
    inc (State (I))
  else
    State (I) :=N
  end if
end procedure

procedure RewardOffUnit (I)
  if State (I) ≠ N + 1 then
    dec (State (I))
  end if
end procedure
```

Figure 10. Survival algorithm.

Table 1. Variances of activation for hidden units.

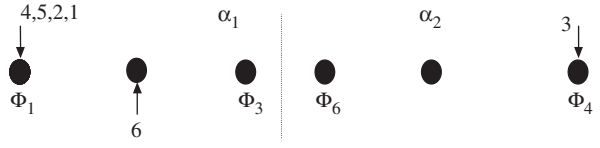
	Variance of unit activation						Widths		Average	
Step	6	5	4	3	2	1	X_{OFF}	X_{ON}	M_{OFF}	M_{ON}
1	0.8	0.3	0.2	0.3	0.4	0.6	0	0.2	0	0.43
2	0.6	0.4	0.3	0.2	0.3	0.6	0	0.15	0	0.38
3	0.3	0.2	0.3	0.2	0.2	0.3	0	0.07	0	0.25
4	0.25	0.35	0.7	0.2	0.6	0.5	0	0.17	0	0.43
5	0.2	0.7	0.3	0.15	0.7	0.7	0.0	0.17	0.15	0.52
6	0.25	0.7	0.25	0.12	0.25	0.7	0.06	0.17	0.12	0.43
7	0.2	0.3	0.4	0.1	0.7	0.7	0.06	0.17	0.15	0.52

Step 3: In this step, units 3 and 4 are penalised, unit 6 is rewarded, and remaining units do not change their states. At the end of this step, states of the automaton and their associated hidden units are given below:

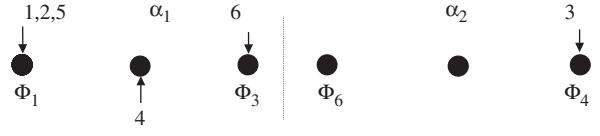


Step 4: In this step, no unit changes state.

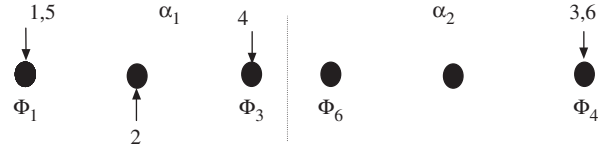
Step 5: In this step, units 3 and 6 are penalised, unit 4 is rewarded, and remaining units do not change their states. At the end of this step, states of the automaton and their associated hidden units are given below:



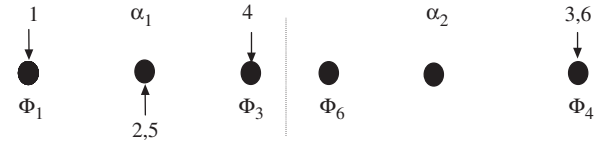
Step 6: In this step, units 4 and 6 are penalised, and remaining units do not change their states. At the end of this step, states of the automaton and their associated hidden units are given below:



Step 7: In this step, units 2, 4, and 6 are penalised, and remaining units do not change their states. At the end of this step, states of the automaton and their associated hidden units are given below:



Step 8: In this step, unit 5 is penalised, and remaining units do not change their states. At the end of this step, states of the automaton and their associated hidden units are given below:



6. Simulation results

In order to evaluate the performance of the proposed algorithm simulations are carried out on six different learning problems: English digit recognition, three bit parity, encoding, symmetry, XOR and Monk III problems for some of which the minimum number of hidden units are known. Results obtained are compared with results of two existing algorithms called iterative pruning (Castellano et al. 1997) and conversational pruning (Sietsma and Dow 1991) algorithms. Results of simulations, which are summarised in Tables 2–8, show the superiority of the survival algorithm over the two above mentioned methods. For all simulations we have used HULA (2, 7, 60); i.e. a HULA with 2 actions, depth of 7 and 60 hidden units.

Three-bit odd-parity problem: In this problem a string of three inputs is applied to the network, the output of the network is zero (one) if the number of

Table 2. Simulation results for 3-bit parity problem.

Algorithm	Survival algorithm			Conversational pruning algorithm			Iterative pruning algorithm		
Network	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate
1	3	0.004859	100	4	0.0275	100	4	0.0294	100
2	4	0.004919	100	4	0.0275	100	3	0.0809	100
3	3	0.004979	100	4	0.0262	100	4	0.0236	100
4	3	0.004984	100	5	0.0338	100	4	0.0523	100
5	4	0.004887	100	3	0.0379	100	4	0.0236	100
6	4	0.004849	100	4	0.0205	100	3	0.0297	100
7	4	0.004731	100	3	0.0213	100	3	0.0879	100
8	3	0.004778	100	4	0.0428	100	4	0.0258	100
9	4	0.004762	100	2	0.0193	100	4	0.0253	100
10	3	0.004993	100	3	0.0379	100	3	0.0894	100
Average	3.5	0.004584	100	3.555	0.0296	100	3.555	0.0487	100

Table 3. Simulation results for encoding problem.

Algorithm	Survival algorithm			Conversational pruning algorithm			Iterative pruning algorithm		
Network	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate
1	3	0.02193	100	7	0.0660	100.0	4	0.0962	100.0
2	3	0.00998	100	7	0.0660	100.0	4	0.0626	100.0
3	2	0.12366	100	5	0.0992	100.0	5	0.0394	100.0
4	3	0.00993	100	5	0.0992	100.0	4	0.0959	100.0
5	3	0.00996	100	4	0.0955	100.0	4	0.0404	100.0
6	3	0.00995	100	5	0.1333	100.0	5	0.0531	100.0
7	3	0.02190	100	6	0.0724	100.0	5	0.0531	100.0
8	3	0.00998	100	7	0.0650	100.0	4	0.1188	100.0
9	3	0.00998	100	5	0.0845	100.0	3	0.1268	100.0
10	3	0.00999	100	5	0.0845	100.0	4	0.0827	100.0
Average	2.9	0.01856	100	5.6	0.08656	100	4.2	0.0769	100

Table 4. Simulation results for English digit problem.

Algorithm	Survival algorithm			Conversational pruning algorithm			Iterative pruning algorithm		
Network	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate
1	5	0.04927	90	6	0.1205	90	5	0.0980	100
2	5	0.04949	90	6	0.1205	90	5	0.0980	100
3	5	0.04858	90	6	0.1205	90	5	0.0980	100
4	5	0.04982	90	5	0.2311	90	5	0.0980	100
5	5	0.04628	90	5	0.2311	90	4	0.1305	100
6	5	0.04926	90	7	0.1431	90	5	0.0830	100
7	5	0.04809	90	7	0.1431	90	5	0.0830	100
8	5	0.04908	90	8	0.0778	100	5	0.1309	100
9	5	0.04919	90	4	0.2269	90	5	0.1309	100
10	4	0.00497	90	5	0.1385	100	5	0.1309	100
Average	4.9	0.0380	90	5.9	0.15531	92	4.9	0.10812	100

Table 5. Simulation results for XOR problem.

Algorithm	Survival algorithm			Conversational pruning algorithm			Iterative pruning algorithm		
Network	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate
1	2	0.0325	100	4	0.0182	100	4	0.0119	100
2	3	0.0021	100	4	0.0182	100	4	0.0119	100
3	2	0.0135	100	4	0.0182	100	4	0.0119	100
4	4	0.0041	100	3	0.0160	100	3	0.0118	100
5	2	0.0235	100	3	0.0160	100	3	0.0118	100
6	3	0.0042	100	3	0.0160	100	2	0.0198	100
7	2	0.0315	100	2	0.0190	100	4	0.0118	100
8	4	0.0013	100	2	0.0190	100	4	0.0118	100
9	2	0.0338	100	4	0.0191	100	4	0.0118	100
10	3	0.0001	100	4	0.0191	100	3	0.0294	100
Average	2.7	0.01466	100	3.22	0.01784	100	3.44	0.01466	100

Table 6. Simulation results for 4-bit symmetry problem.

Algorithm	Survival algorithm			Conversational pruning algorithm			Iterative pruning algorithm		
Network	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate
1	2	0.0231	100	5	0.0450	100	3	0.0465	100
2	2	0.0132	100	5	0.0450	100	3	0.0465	100
3	3	0.0267	100	4	0.0618	100	3	0.0772	100
4	2	0.0134	100	4	0.0618	100	4	0.0377	100
5	2	0.0141	100	3	0.0756	100	4	0.0377	100
6	2	0.0243	100	5	0.0291	100	3	0.0562	100
7	3	0.0437	100	5	0.0291	100	2	0.0791	100
8	4	0.0303	100	4	0.0441	100	5	0.0275	100
9	2	0.0131	100	4	0.0441	100	4	0.0453	100
10	3	0.0567	100	6	0.0368	100	3	0.0447	100
Average	2.5	0.02586	100	4.5	0.04724	100	3.4	0.04984	100

Table 7. Simulation results for 6-bit symmetry problem.

Algorithm	Survival algorithm			Conversational pruning algorithm			Iterative pruning algorithm		
Network	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate
1	5	0.000003	100	5	0.3312	96.9	5	0.1666	100.0
2	2	0.00113	100	8	0.6119	90.6	3	0.5813	93.8
3	3	0.01087	100	5	0.3107	96.9	3	0.3950	95.3
4	2	0.10579	95.5	5	0.4103	92.2	4	0.3075	98.4
5	2	0.09442	97.5	5	0.2201	100.0	3	0.2505	100.0
6	2	0.09380	87.5	7	0.4470	92.2	2	0.6555	93.8
7	4	0.09779	97.5	3	0.4888	93.8	3	0.1699	100.0
8	3	0.0793	87.5	5	0.4671	93.8	4	0.4376	95.3
9	3	0.09535	92.5	4	0.3970	92.2	3	0.2259	100.0
10	2	0.09375	90.5	6	0.5481	90.6	3	0.1799	100.0
Average	2.8	0.06722	94.85	5.3	0.42322	93.92	3.3	0.33697	97.66

Table 8. Simulation results for Monk III problem.

Algorithm	Survival algorithm			Conversational pruning algorithm			Iterative pruning algorithm		
Network	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate	No. of hidden units	MSE	Recognition rate
1	10	0.0044	79.6	20	0.0048	73.1	19	0.0049	79.4
2	10	0.0048	85.6	19	0.0047	85.2	20	0.0044	82.9
3	12	0.0048	78.5	20	0.0097	77.1	14	0.0046	82.9
4	12	0.0046	85.6	19	0.0072	81	19	0.0054	75.2
5	12	0.0046	85.6	20	0.0044	82.9	19	0.0053	74.8
6	13	0.005	80.3	20	0.0035	82.6	17	0.0053	82.2
7	12	0.0044	81.7	20	0.0076	83.8	16	0.0066	78.7
8	10	0.0047	77.5	20	0.0061	90.3	14	0.0064	79.6
9	12	0.0045	82.2	20	0.0053	75	16	0.0115	83.6
10	14	0.0047	75.5	20	0.0056	75.7	19	0.0043	83.1
Average	11.7	0.00465	81.21	19.8	0.00589	80.67	17.3	0.00587	80.24

ones in the input is odd (even) (Rumelhart et al. 1986). It is known that to produce n-bit parity, a three layer neural network, which uses Backpropagation algorithm, requires at least n hidden units (Rumelhart et al. 1986). Of course neural networks with other topology can solve this problem using only two hidden units. Survival algorithm is tested on 10 different networks with initial random weights and the result is given in Table 2. For all simulations, survival algorithm has been able to train the network. Figures 11 and 12 show how the number of hidden units and the mean square error change during training for a typical simulation.

***Encoding problem*:** In this problem, a set of orthogonal input patterns are mapped to a set of orthogonal output patterns. It is known that to produce n-bit encoding, a three layer neural network, which uses backpropagation algorithm, requires at least $\log_2 n$ hidden units (Rumelhart et al. 1986). Survival algorithm is tested on 10 different networks with initial random weights and the result is given in Table 3. For all simulations survival algorithm has been able to train the network. Figures 13 and 14 show how the number of hidden units and the mean square error change during training for a typical simulation.

***English digit recognition*:** There are numbers 0 through 9, and each represented by an 8×8 grid of black and white dot as shown in Figure 15.

The network must learn to distinguish these numbers. It is known that to solve this problem, a three layer neural network, which uses backpropagation algorithm, requires at least 4 hidden units (Sperduti and Starita 1993). The proposed algorithm is tested on 10 networks with initial random weights and the results are presented in Table 4. For all simulations survival algorithm has been able to train the network. Figures 16 and 17 show how the number of hidden units and the mean square error change during training for a typical simulation when using HULA (Frean 1990; Arai 1993; Beigy and Meybodi 2004).

XOR problem: It is known that to solve this problem, a three layer neural network, which uses backpropagation algorithm, requires at least 2 hidden units (Rumelhart et al. 1986). The proposed algorithm is tested on 10 networks with initial random weights and the results are presented in Table 5. For all simulations survival algorithm has been able to train the network. Figures 18 and 19 show how the number of hidden units and the mean square error change during training for a typical simulation when using HULA (Frean 1990; Arai 1993; Beigy and Meybodi 2004).

Symmetry problem: This problem classifies input string as to whether or not they are symmetric about center. It is known that to solve this problem, a three layer neural network, which uses backpropagation algorithm, requires at least 2 hidden units (Rumelhart et al. 1986). Two sets of simulations are conducted, one for 4-bit symmetry problem, and another for 6-bit Symmetry Problem. Ten networks are tested for each problem and results are given in Tables 6 and 7. For all simulations survival algorithm has been able to train the network. Figures 20–23 show how the number of hidden units and the mean square error change during training for a typical simulation when using HULA (Frean 1990; Arai 1993; Beigy and Meybodi, 2004).

Note that for XOR and Symmetry Problem as the number of hidden units gets closer to the optimal number the rate of convergence of the network decreases.

Monk problems: The Monk's problems are a wide benchmark for comparing classification algorithms (Thrun and Al 1991). The Monk's problems relay on an artificial robot domain, in which robot domains are described by six different attributes. The learning task

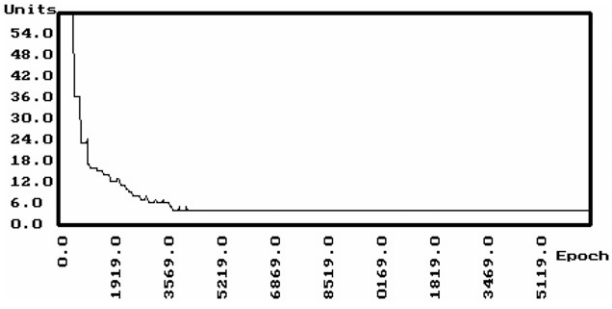


Figure 11. The number of hidden units.

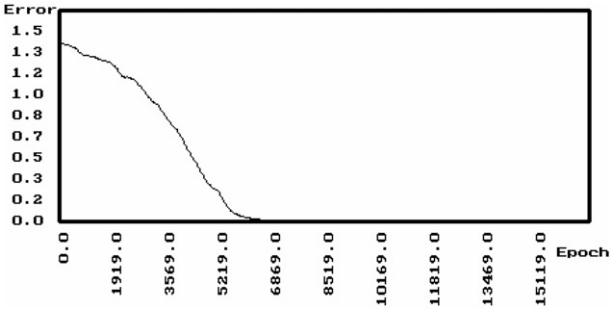


Figure 12. The mean square error change.

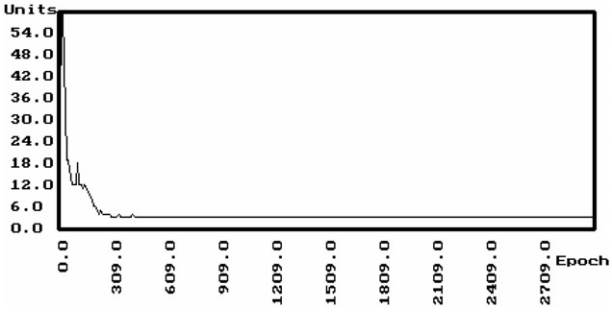


Figure 13. The number of hidden units.

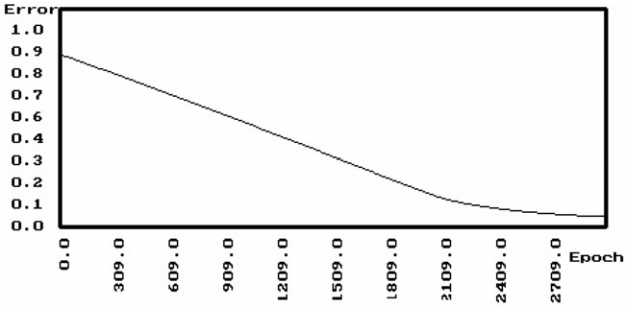


Figure 14. The mean square error change.

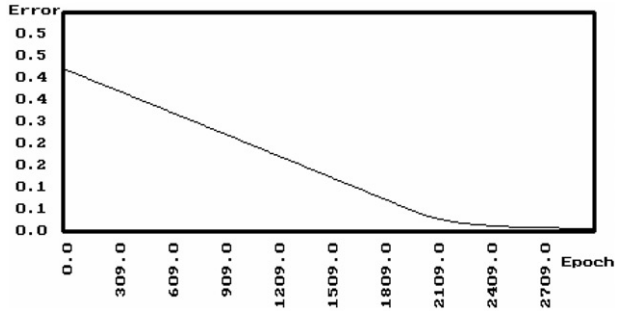


Figure 16. The number of hidden units.

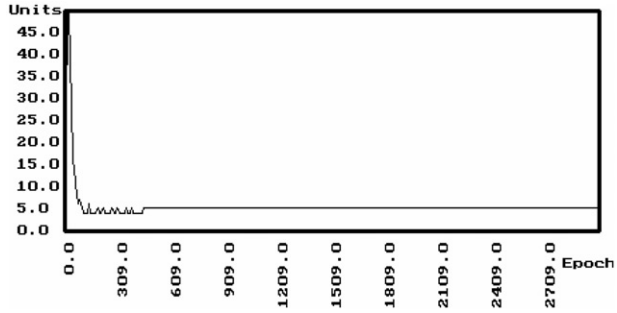


Figure 17. The mean square error change.

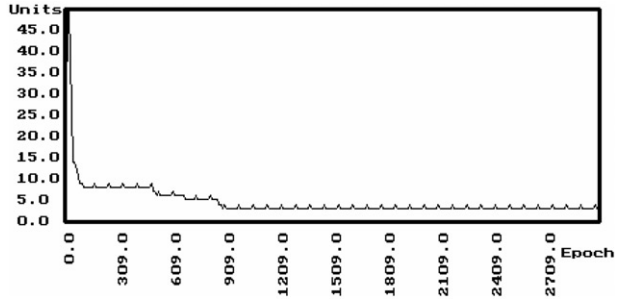


Figure 18. The number of hidden units.

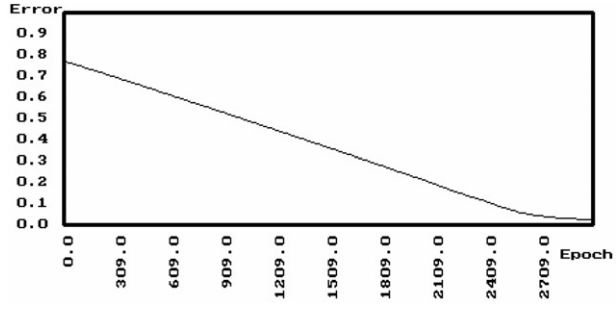


Figure 19. The mean square error change.

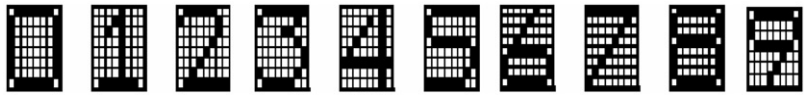


Figure 15. The training patterns for English digit recognition.

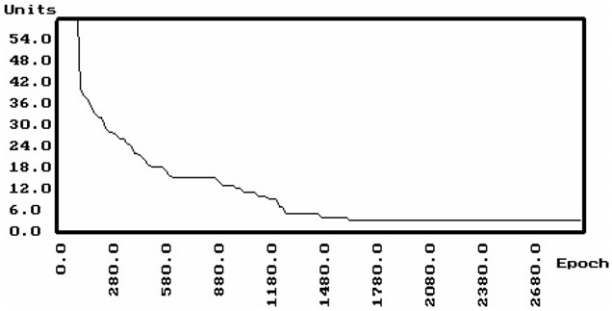


Figure 20. The number of hidden units for 4-bit symmetry.

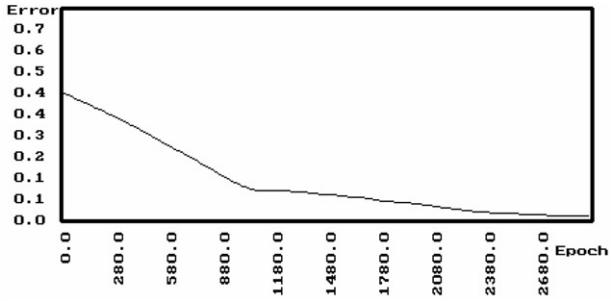


Figure 21. The mean square error change for 4-bit symmetry.

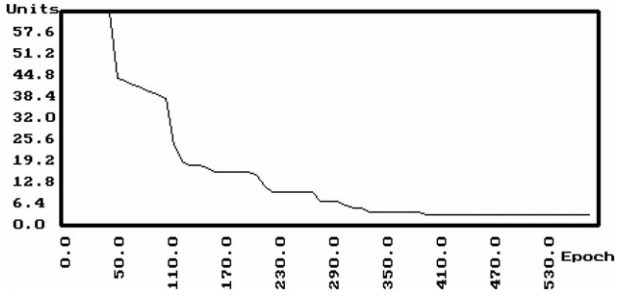


Figure 22. The number of hidden units for 6-bit symmetry.

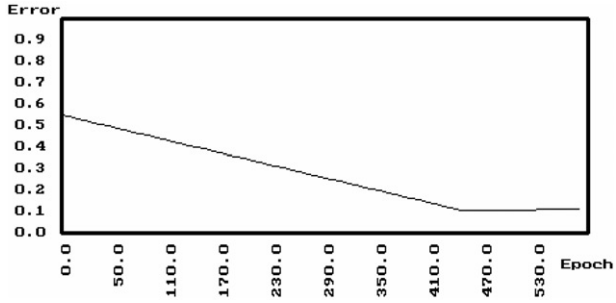


Figure 23. The mean square error change for 6-bit symmetry.

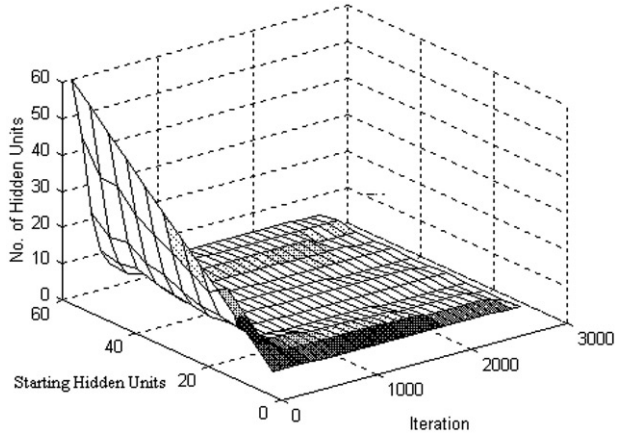


Figure 24. Independency of survival algorithm with respect to initial neurons.

Table 9. The average running time of algorithms.

Survival algorithm	Iterative pruning algorithm	Conversational pruning algorithm
1458 ms	2841 ms	2041 ms

is a binary classification task. Each problem is given by a logical description of a class. Robots belong either to this class or not, but instead of providing a complete class description to the learning problem. The Monk III problem has 5% classification noise. The proposed algorithm is tested on 10 networks with initial random weights and the results are presented in Table 8. For all simulations survival algorithm has been able to train the network.

Remark 1: Simulation results show that networks produced by the survival algorithm are smaller than the networks produced by both conversational and iterative pruning algorithms.

Remark 2: The survival algorithm is faster than both conversational and iterative pruning algorithms. Table 9 shows the average of the actual running time taken for different runs for Monk III problem for all three algorithms. All algorithms are run a Pentium III 2.4 MHZ workstation.

Remark 3: The survival algorithm finds a near optimal number of hidden units independent of initial number of hidden units. Figure 24 shows how the initial number of hidden units affects the final topology for English digit recognition problem. This plot indicates the fact that the final topology produced by the proposed algorithm is independent of the

number of hidden units with which the algorithm begins. Each point of the plot is averaged over 10 different runs

7. Conclusions

In this article, we first introduced a learning automaton and studied its behaviour. Then an algorithm based on the proposed learning automaton, called survival algorithm, for determination of the number of hidden units of three layers neural networks was proposed. The proposed algorithm uses learning automata as a global search method in order to increase the probability of obtaining the optimal topology. Survival algorithm starts with a large network, and then by adding and deleting hidden units obtains a topology very close to the optimal topology. It is shown through simulation that the final topology is independent of the starting number of the hidden units considered by the algorithm.

Acknowledgement

The authors thank the reviewers for their very helpful comments and suggestions.

Notes on contributors

Hamid Beigy received the BS and MS degrees in Computer Engineering from Shiraz University in Iran, in 1992 and 1995, respectively. He also received the PhD degree in Computer Engineering from Amirkabir University of Technology in Iran, in 2004. Currently, he is an Assistant Professor in Computer Engineering Department at Sharif University of Technology, Tehran, Iran. His research interests include channel management in cellular networks, learning systems, parallel algorithms and soft computing.

Mohammad Reza Meybodi received the BS and MS degrees in Economics from Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received the MS and PhD degree from Oklahoma University, USA, in 1980 and 1983, respectively in Computer Science. Currently, he is a full professor in Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to his current position, he worked from 1983 to 1985 as an assistant professor at Western Michigan University, and from 1985 to 1991 as an associate professor at Ohio University, USA. His research interests include channel management in cellular networks, learning systems, parallel algorithms, soft computing and software development.

References

Adibi, P., Meybodi, M.R., and Safabaksh, R. (2005), "Unsupervised Learning of Synaptic Delays Based on Learning Automata in an Rbf-Like Network of Spiking Neurons for Data Clustering," *Journal of Neurocomputing*, 64, 335–337.

Angeline, P.J., Saunders, G.M., and Pollack, J.B. (1994), "Evolutionary Algorithm that Construct Recurrent Neural Networks," *IEEE Transactions on Neural Networks*, 5(1), 54–65.

Arai, M. (1993), "Bounds on the Number of Hidden Units in Binary-Valued Three-Layer Neural Networks," *Neural Networks*, 6, 855–860.

Beigy, H., and Meybodi, M.R. (1998a), "A Fast Method for Determining the Number of Hidden Units in Feedforward Neural Networks," *Proceedings of CSICC-97*, *Tehran, Iran*, 1, 414–420 (In Persian).

——— (1998b), *Optimization of Topology of Neural Networks*: *A Survey*, Technical Reports, Computer Eng. Dept. Amirkabir University of Technology, Tehran, Iran.

——— (2000), "Solving the Graph Isomorphism Problem Using Learning Automata," in *Proceedings of 5th Annual Int. Computer Society of Iran Computer Conference*, CISCC-2000, pp. 402–415.

(2001a), "Backpropagation Algorithm Adaptation Parameters Using Learning Automata," *International Journal of Neural Systems*, 11(3), 219–228.

——— (2001b), "Adaptation of Momentum Factor and Steepness Parameter in Backpropagation Using Fixed Structure Learning Automata," *International Journal of Science and Technology* (*Iranica Saientia*), 8(4), 250–264.

——— (2002a), "Call Admission in Cellular Networks: A Learning Automata Approach," *Springer-Verlag Lecture Notes in Computer science*, Vol. 2510, New York: Springer-Verlag, pp. 450–457.

——— (2002b), "Learning Automata Based Dynamic Guard Channel Scheme," in *Springer-Verlag Lecture Notes in Computer Science* (Vol. 2510), New York: Springer-Verlag, pp. 643–650.

——— (2003), "An Adaptive Uniform Fractional Guard Channel Algorithm: A Learning Automata Approach," in *Springer-Verlag Lecture Notes in Computer Science* (Vol. 2690), New York: Springer-Verlag, pp. 405–409.

——— (2004), "Adaptive Uniform Fractional Channel Algorithms," *Iranian Journal of Electrical and Computer Engineering*, 3, 47–53.

——— (2005), "An Adaptive Call Admission Algorithm for Cellular Networks," *Journal of Computer and Electrical Engineering*, 31(2), 132–151.

Beigy, H., Meybodi, M.R., and Menhaj, M.B. (2002), "Utilization of Fixed Structure Learning Automata for Adaptation of Learning Rate in Backpropagation Algorithm," *Pakestanian Journal of Applied Science*, 2(4), 437–444.

Castellano, G., Fanelli, A.M., and Pelillo, M. (1997), "A Iterative Pruning Algorithm for Feedforward Neural

Networks,'' *IEEE Transactions on Neural Networks*, 8(3), 519–531.

Fahlman, S.E., and Lebier, C. (1990), ''The Cascade-Correlation Learning Architecture,'' *Advances in Neural Information Processing System*, 2, 524–532.

Frean, M. (1990), ''The Upstart: A Method for Constructing and Training Feedforward Neural Networks,'' *Neural Computation*, 2(2), 198–209.

Hashim, A.A., Amir, S., and Mars, P. (1986), in *Adaptive and Learning Systems*, ed. K.S. Narendra, New York: Plenum Press, pp. 229–234.

Hirose, Y., Yamashita, K., and Hijya, S. (1991), ''Back-Propagation Algorithm Which Varies the Number of Hidden Units,'' *Neural Networks*, 4(1), 61–66.

Huang, S.C., and Huang, Y.F. (1991), ''Bounds on the Number Hidden Neurons in Multilayer Preceptrons,'' *IEEE Transactions on Neural Networks*, 2(1), 47–56.

Judd, J.S. (1990), *Neural Network Design and the Time Complexity of Learning*, MA: MIT Press.

Kruschke, J.H. (1988), ''Creating Local and Distributed Bottelnecks in Hidden Layer of Backpropagation Networks'' in *Proc. of Connectionist Models, Summer School*, eds. D. Tourestzky, G. Hinton, & T. Sejnowski, pp. 120–126.

——— (1989), ''Improving Generalization in Backpropagation Networks,'' *Proc. of Int. Joint Conf. on Neural Networks*, 1, 443–447.

Kwok, T.Y., and Yeung, D.Y. (1997), ''Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems,'' *IEEE Transactions on Neural Networks*, 8(3), 630–645.

Lin, J.H., and Vitter, J.S. (1991), ''Complexity Results on Learning by Neural Nets,'' *Machine Learning*, 6, 211–230.

Maniezzo, V. (1994), ''Genetic Evolution of the Topology and Weight Distribution of Neural Networks,'' *IEEE Transactions on Neural Networks*, 5(1), 39–53.

Marchand, M., Golea, M., and Rujan, R. (1990), ''A Convergence Theorem for Sequential Learning in Two-Layer Perceptrons,'' *Europhysics Letters*, 11, 487–492.

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

Mars. P, Narendra. K.S., and Chrystall, M. (1983), ''Learning Automata Control of Computer Communication Networks,'' in *Proceedings of Third Yale Workshop on Applications of Adaptive Systems Theory*, Yale University.

Meltser, M., Shoham, M., and Manevitz, L.M. (1996), ''Approximating Function by Neural Networks: A Constructive Solution in the Uniform Norm,'' *Neural Networks*, 9(6), 965–978.

Meybodi, M.R., and Beigy, H. (2002a), ''A Note on Learning Automata Based Schemes for Adaptation of Bp Parameters,'' *Journal of Neurocomputing*, 48, 957–974.

——— (2002b), ''New Class of Learning Automata Based Scheme for Adaptation of Backpropagation Algorithm Parameters,'' *International Journal of Neural Systems*, 12(1), 45–68.

Meybodi, M.R., and Lakshmivarhan, S. (1983), ''A Learning Approach to Priority Assignment in a Two Class M/M/1 Queuing System with Unknown Parameters,'' in *Proceedings of Third Yale Workshop on Applications of Adaptive Systems Theory*, Yale University, pp. 106–109.

Mezard, M., and Nadal, J.P. (1989), ''Learning in Feedforward Neural Networks: The Tiling Algorithm,'' *Journal of Physics*, 22(12), 1285–1296.

Minor, J.M. (1993), ''Parity with Two Layer Feedforward Nets,'' *Neural Networks*, 6(5), 705–707.

Nabhan, T.M., and Zomaya, A.Y. (1994), ''Toward Neural Networks Structures for Function Approximation,'' *Neural Networks*, 7(1), 89–99.

Narendra, K.S., and Thathachar, M.A.L. (1989), *Learning Automata: An Introduction*, Englewood Cliffs: Prentice-Hall.

Oommen, B.J., and Croix, E.V. de St (1996), ''Graph Partioning Using Learning Automata,'' *IEEE Transactions on Computers*, 45(2), 195–208.

Oommen, B.J., and Ma, D.C.Y. (1988), ''Deterministic Learning Automata Solutions to the Equipartitioning Problem,'' *IEEE Transactions on Computers*, 37(1), 2–13.

Oommen, B.J., Valiveti, R.S., and Zgierski, J.R. (1991), ''An Adaptive Learning Solution to the Keyboard Optimization Problem,'' *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), 1608–1618.

Reed, R. (1993), ''Pruning Algorithms – A Survey,'' *IEEE Transactions on Neural Networks*, 4(5), 740–747.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986), ''Learning Internal Representations by Error Backropagation'' *Parallel Distributed Processing*, Cambridge, MA: MIT Press.

Schaffer, J.D., Whitely, D., and Eshelman, L.J. (1992), ''Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art,'' *IEEE Proceedings COGANN-92*, 1, 1–37.

Sietsma, J., and Dow, R.J.F. (1991), ''Creating Artifitial Neural Networks that Generalize,'' *Neural Networks*, 4(1), 67–79.

Sirat, J.A., and Nadal, J.P. (1990), ''Neural Trees: A New Tool for Classification,'' Preprint, *Laboratories d'Electronique, Philips*, Limeil Brevannes, France.

Sperduti, A., and Starita, A. (1993), ''Speed Up Learning and Network Optimization with Extended Backpropagation,'' *Neural Networks*, 6, 365–383.

Tamura, S., and Tateishi, M. (1997), ''Capabilities of a Four-Layered Feedforward Neural Network: Four Layers Versus Three,'' *IEEE Transactions on Neural Networks*, 8(2), 251–255.

Thathachar, M.A.L., and Sastry, P.S. (1987), ''Learning Optimal Discriminant Functions Through a Cooperative Game of Automata,'' *IEEE Transaction Systems, Man and Cybernetics*, SMC-27, 73–85.

Thrun, S.B., Bala, T., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Džeroski, S., Fahlman, S.E., Fisher, D., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, T., Michalski, R.S., Mitchell, T., Pachowicz, P., Reich, Y., Vafaie, H., Welde, W. Van de, Wenzel, W., Wnek, J., and Zhang, J. (1991), ''The Monk's Problems: A Performance Comparison of Different Learning Algorithms,'' Technical Report: CMU-CS-91-197, Carnegie Mellon University.

Whitley, D., and Bogart, C. (1990), "The Evolution of Connectivity: Pruning Neural Networks Using Genetic Algorithms," *Proceedings of Int. Joint Conf. on Neural Networks*, I, 134.

Yao, X., and Liu, Y. (1997), "A New Evolutionary System for Evolving Artificial Neural Networks," *IEEE Transactions on Neural Networks*, 8(3), 694–713.

Yeung, D.Y. (1991), "Automatic Determination of Network Size for Supervised Learning," *IEEE International Joint Conference on Neural Networks*, 1, 158–164.

Yu, X.H. (1992), "Can Backpropagation Error Surface not have Local Minima," *IEEE Transactions on Neural Networks*, 3(6), 1019–1021.