Tuning of the Structure and Parameters of a Neural Network Using an Improved Genetic Algorithm

Frank H. F. Leung, *Member, IEEE*, H. K. Lam, S. H. Ling, and Peter K. S. Tam

***Abstract*—This paper presents the tuning of the structure and parameters of a neural network using an improved genetic algorithm (GA). It will also be shown that the improved GA performs better than the standard GA based on some benchmark test functions. A neural network with switches introduced to its links is proposed. By doing this, the proposed neural network can learn both the input–output relationships of an application and the network structure using the improved GA. The number of hidden nodes is chosen manually by increasing it from a small number until the learning performance in terms of fitness value is good enough. Application examples on sunspot forecasting and associative memory are given to show the merits of the improved GA and the proposed neural network.**

***Index Terms*—Genetic algorithm (GA), neural networks, parameter learning, structure learning.**

I. Introduction

GENETIC algorithm (GA) is a directed random search technique [1] that is widely applied in optimization problems [1], [2], [5]. This is especially useful for complex optimization problems where the number of parameters is large and the analytical solutions are difficult to obtain. GA can help to find out the optimal solution globally over a domain [1], [2], [5]. It has been applied in different areas such as fuzzy control [9]–[11], [15], path planning [12], greenhouse climate control [13], modeling and classification [14] etc.

A lot of research efforts have been spent to improve the performance of GA. Different selection schemes and genetic operators have been proposed. Selection schemes such as rank-based selection, elitist strategies, steady-state election and tournament selection have been reported [32]. There are two kinds of genetic operators, namely crossover and mutation. Apart from random mutation and crossover, other crossover and mutation mechanisms have been proposed. For crossover mechanisms, two-point crossover, multipoint crossover, arithmetic crossover, and heuristic crossover have been reported [1], [31]–[33]. For mutation mechanisms, boundary mutation, uniform mutation, and nonuniform mutation can be found [1], [31]–[33].

Neural network was proved to be a universal approximator [16]. A three-layer feedforward neural network can approximate any nonlinear continuous function to an arbitrary accuracy. Neural networks are widely applied in areas such as prediction [7], system modeling, and control [16]. Owing to its particular structure, a neural network is very good in learning [2] using some learning algorithms such as GA [1] and backpropagation [2]. In general, the learning steps of a neural network are as follows. First, a network structure is defined with a fixed number of inputs, hidden nodes and outputs. Second, an algorithm is chosen to realize the learning process. However, a fixed structure may not provide the optimal performance within a given training period. A small network may not provide good performance owing to its limited information processing power. A large network, on the other hand, may have some of its connections redundant [18], [19]. Moreover, the implementation cost for a large network is high. To obtain the network structure automatically, constructive and destructive algorithms can be used [18]. The constructive algorithm starts with a small network. Hidden layers, nodes, and connections are added to expand the network dynamically [19]–[24]. The destructive algorithm starts with a large network. Hidden layers, nodes, and connections are then deleted to contract the network dynamically [25], [26]. The design of a network structure can be formulated into a search problem. GAs [27], [28] were employed to obtain the solution. Pattern-classification approaches [29] can also be found to design the network structure. Some other methods were proposed to learn both the network structure and connection weights. An ANNA ELEONORA algorithm was proposed [36]. New genetic operator and encoding procedures that allows an opportune length of the coding string were introduced. Each gene consists of two parts: the connectivity bits and the connection weight bits. The former indicates the absence or presence of a link, and the latter indicates the value of the weight of a link. A GNARL algorithm was also proposed in [37]. The number of hidden nodes and connection links for each network is first randomly chosen within some defined ranges. Three steps are then used to generate an offspring: copying the parents, determining the mutations to be performed, and mutating the copy. The mutation of a copy is separated into two classes: the parametric mutations that alter the connection weights, and the structural mutations that alter the number of hidden nodes and the presence of network links. An evolutionary system named EPNet can also be found for evolving neural networks [19]. Rank-based selection and five mutations were employed to modify the network structure and connection weights.

In this paper, a three-layer neural network with switches introduced in some links is proposed to facilitate the tuning of the network structure in a simple manner. A given fully connected feedforward neural network may become a partially connected network after learning. This implies that the cost of implementing the proposed neural network, in terms of hardware and processing time, can be reduced. The network structure and

Manuscript received September 5, 2001; revised February 6, 2002 and July 2, 2002. This work was supported by the Centre for Multimedia Signal Processing, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, under Project A432.

The authors are with the Centre for Multimedia Signal Processing, Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Kowloon, Hong Kong.

Digital Object Identifier 10.1109/TNN.2002.804317

```
Procedure of the standard GA
begin
        $\tau \leftarrow 0$          // $\tau$: number of iteration
initialize $\mathbf{P}(\tau)$          //$\mathbf{P}(\tau)$: population for iteration $\tau$
        evaluate $f(\mathbf{P}(\tau))$          // $f(\mathbf{P}(\tau))$:fitness function
while (not termination condition) do
    begin
        $\tau \leftarrow \tau+1$
        select 2 parents $\mathbf{p}_1$ and $\mathbf{p}_2$ from $\mathbf{P}(\tau-1)$
        perform genetic operations (crossover and mutation)
        reproduce a new $\mathbf{P}(\tau)$
        evaluate $f(\mathbf{P}(\tau))$
    end
end
```

Fig. 1. Procedure of standard GA.

parameters will be tuned simultaneously using a proposed improved GA. As application examples, the proposed neural network with link switches tuned by the improved GA is used to estimate the number of sunspots [7], [8] and realize an associative memory. The results will be compared with those obtained by traditional feedforward networks [2] trained by 1) the standard GA with arithmetic crossover and nonuniform mutation [1], [2], [5] and 2) the backpropagation with momentum and adaptive learning rate [30].

This paper is organized as follows. In Section II, the improved genetic algorithm is presented. In Section III, it will be shown that the improved GA performs more efficiently than the standard GA [1], [2], [5] based on some benchmark test functions [3], [4], [6], [17]. In Section IV, the neural network with link switches, and the tuning of its structure and parameters using the improved GA will be presented. Application examples will be presented in Section V. A conclusion will be drawn in Section VI.

II. Improved GA

The standard GA process [1], [2], [5] is shown in Fig. 1. First, a population of chromosomes is created. Second, the chromosomes are evaluated by a defined fitness function. Third, some of the chromosomes are selected for performing genetic operations. Forth, genetic operations of crossover and mutation are performed. The produced offspring replace their parents in the initial population. In this reproduction process, only the selected parents in the third step will be replaced by their corresponding offspring. This GA process repeats until a user-defined criterion is reached. In this paper, the standard GA is modified and new genetic operators are introduced to improve its performance. The improved GA process is shown in Fig. 2. Its details will be given as follows.

A. Initial Population

The initial population is a potential solution set P. The first set of population is usually generated randomly

$$P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{pop\_size}\} \tag{1}$$

$$\mathbf{p}_i = [p_{i_1} \quad p_{i_2} \quad \cdots \quad p_{i_j} \quad \cdots \quad p_{i_{no\_vars}}]$$
$$i = 1, 2, \ldots, pop\_size;$$
$$j = 1, 2, \ldots, no\_vars \tag{2}$$

$$para^j_{\min} \le p_{i_j} \le para^j_{\max} \tag{3}$$

where pop_size denotes the population size; no_vars denotes the number of variables to be tuned; p_{i_j}, $i = 1, 2, \ldots, pop\_size$; $j = 1, 2, \ldots, no\_vars$, are the parameters to be tuned; $para^j_{\min}$ and $para^j_{\max}$ are the minimum and maximum values of the parameter p_{i_j}, respectively for all i. It can be seen from (1)–(3) that the potential solution set P contains some candidate solutions $\mathbf{p}_i$ (chromosomes). The chromosome $\mathbf{p}_i$ contains some variables p_{i_j} (genes).

B. Evaluation

Each chromosome in the population will be evaluated by a defined fitness function. The better chromosomes will return higher values in this process. The fitness function to evaluate a chromosome in the population can be written as

$$\text{fitness} = f(\mathbf{p}_i). \tag{4}$$

The form of the fitness function depends on the application.

C. Selection

Two chromosomes in the population will be selected to undergo genetic operations for reproduction by the method of spinning the roulette wheel [1]. It is believed that high potential parents will produce better offspring (survival of the best ones). The chromosome having a higher fitness value should therefore have a higher chance to be selected. The selection can be done by assigning a probability q_i to the chromosome $\mathbf{p}_i$

$$q_i = \frac{f(\mathbf{p}_i)}{\sum_{k=1}^{pop\_size} f(\mathbf{p}_k)}, \qquad i = 1, 2, \ldots, pop\_size. \tag{5}$$

```
Procedure of the improved GA
begin
        $\tau \leftarrow 0$          // $\tau$: number of iteration
        initialize $\mathbf{P}(\tau)$                  //$\mathbf{P}(\tau)$: population for iteration $\tau$
        evaluate $f(\mathbf{P}(\tau))$      // $f(\mathbf{P}(\tau))$:fitness function
while (not termination condition) do
        begin
        $\tau \leftarrow \tau+1$
        select 2 parents $\mathbf{p}_1$ and $\mathbf{p}_2$ from $\mathbf{P}(\tau-1)$
        perform crossover operation according to equations (7) to (13)
        perform mutation operation according to equation (14) to generate three
        offspring $\mathbf{nos}_1$, $\mathbf{nos}_2$ and $\mathbf{nos}_3$
        // reproduce a new $\mathbf{P}(\tau)$
            if random number < $p_a$  // $p_a$: probability of acceptance
                The one among $\mathbf{nos}_1$, $\mathbf{nos}_2$ and $\mathbf{nos}_3$ with the largest fitness value
                replaces the chromosome with the smallest fitness value in the
                population
            else begin
                if $f(\mathbf{nos}_1)$ > smallest fitness value in the $\mathbf{P}(\tau-1)$
                    $\mathbf{nos}_1$ replaces the chromosome with the smallest fitness value
                end
                if $f(\mathbf{nos}_2)$ > smallest fitness value in the updated $\mathbf{P}(\tau-1)$
                    $\mathbf{nos}_2$ replaces the chromosome with the smallest fitness value
                end
                if $f(\mathbf{nos}_3)$ > smallest fitness value in the updated $\mathbf{P}(\tau-1)$
                    $\mathbf{nos}_3$ replaces the chromosome with the smallest fitness value
                end
            end
            evaluate $f(\mathbf{P}(\tau))$
        end
end
```

Fig. 2. Procedure of the improved GA.

The cumulative probability $\hat{q}_i$ for the chromosome $\mathbf{p}_i$ is defined as

$$\hat{q}_i = \sum_{k=1}^{i} q_k, \qquad i = 1, 2, \ldots, pop\_size. \tag{6}$$

The selection process starts by randomly generating a nonzero floating-point number, $d \in [0 \quad 1]$. Then, the chromosome $\mathbf{p}_i$ is chosen if $\hat{q}_{i-1} < d \leq \hat{q}_i$ ($\hat{q}_0 = 0$). It can be observed from this selection process that a chromosome having a larger $f(\mathbf{p}_i)$ will have a higher chance to be selected. Consequently, the best chromosomes will get more offspring, the average will stay and the worst will die off. In the selection process, only two chromosomes will be selected to undergo the genetic operations.

D. Genetic Operations

The genetic operations are to generate some new chromosomes (offspring) from their parents after the selection process. They include the crossover and the mutation operations.

1) Crossover: The crossover operation is mainly for exchanging information from the two parents, chromosomes $\mathbf{p}_1$ and $\mathbf{p}_2$, obtained in the selection process. The two parents will produce one offspring. First, four chromosomes will be generated according to the following mechanisms:

$$\mathbf{os}_c^1 = [os_1^1 \quad os_2^1 \quad \cdots \quad os_{no\_vars}^1] = \frac{\mathbf{p}_1 + \mathbf{p}_2}{2} \tag{7}$$

$$\begin{aligned}\mathbf{os}_c^2 &= [os_1^2 \quad os_2^2 \quad \cdots \quad os_{no\_vars}^2] \\ &= \mathbf{p}_{\max}(1-w) + \max(\mathbf{p}_1, \mathbf{p}_2)\, w\end{aligned} \tag{8}$$

$$\begin{aligned}\mathbf{os}_c^3 &= [os_1^3 \quad os_2^3 \quad \cdots \quad os_{no\_vars}^3] \\ &= \mathbf{p}_{\min}(1-w) + \min(\mathbf{p}_1, \mathbf{p}_2)\, w\end{aligned} \tag{9}$$

$$\begin{aligned}\mathbf{os}_c^4 &= [os_1^4 \quad os_2^4 \quad \cdots \quad os_{no\_vars}^4] \\ &= \frac{(\mathbf{p}_{\max} + \mathbf{p}_{\min})(1-w) + (\mathbf{p}_1 + \mathbf{p}_2)w}{2}\end{aligned} \tag{10}$$

$$\mathbf{p}_{\max} = [para_{\max}^1 \quad para_{\max}^2 \quad \cdots \quad para_{\max}^{no\_vars}] \tag{11}$$

$$\mathbf{p}_{\min} = [para_{\min}^1 \quad para_{\min}^2 \quad \cdots \quad para_{\min}^{no\_vars}] \tag{12}$$

where $w \in [0 \quad 1]$ denotes the weight to be determined by users, $\max(\mathbf{p}_1, \mathbf{p}_2)$ denotes the vector with each element obtained by taking the maximum among the corresponding element of $\mathbf{p}_1$ and $\mathbf{p}_2$. For instance, $\max([1 \quad -2 \quad 3], [2 \quad 3 \quad 1]) = [2 \quad 3 \quad 3]$. Similarly, $\min(\mathbf{p}_1, \mathbf{p}_2)$ gives a vector by taking the minimum value. For instance, $\min([1 \quad -2 \quad 3], [2 \quad 3 \quad 1]) = [1 \quad -2 \quad 1]$. Among $\mathbf{os}_c^1$ to $\mathbf{os}_c^4$, the one with the largest fitness value is used as the offspring of the crossover operation. The offspring is defined as

$$\mathbf{os} \equiv [\, os_1 \quad os_2 \quad \cdots \quad os_{no\_vars} \,] = \mathbf{os}_c^{i_{os}}. \tag{13}$$

i_{os} denotes the index i which gives a maximum value of $f(\mathbf{os}_c^i)$, $i = 1, 2, 3, 4$.

If the crossover operation can provide a good offspring, a higher fitness value can be reached in less iteration. In general, two-point crossover, multipoint crossover, arithmetic crossover or heuristic crossover can be employed to realize the crossover operation [1], [31]–[33]. The offspring generated by these methods, however, may not be better than that from our approach. As seen from (7)–(10), the potential offspring spreads over the domain. While (7) and (10) result in searching around the center region of the domain [a value of w near to one in (10) can move $\mathbf{os}_c^4$ to be near $(\mathbf{p}_1 + \mathbf{p}_2)/2$], (8) and (9) move the potential offspring to be near the domain boundary [a large value of w in (8) and (9) can move $\mathbf{os}_c^2$ and $\mathbf{os}_c^3$ to be near $\mathbf{p}_{\max}$ and $\mathbf{p}_{\min}$ respectively].

2) Mutation: The offspring (13) will then undergo the mutation operation. The mutation operation is to change the genes of the chromosomes. Consequently, the features of the chromosomes inherited from their parents can be changed. Three new offspring will be generated by the mutation operation

$$\begin{aligned}\mathbf{nos}_j = {} & [os_1 \quad os_2 \ \cdots \ os_{no\_vars}] \\ & + [b_1 \Delta nos_1 \quad b_2 \Delta nos_2 \ \cdots \ b_{no\_vars} \Delta nos_{no\_vars}] \\ & j = 1, 2, 3 \end{aligned} \tag{14}$$

where $b_i, i = 1, 2, \ldots, no\_vars$, can only take the value of 0 or 1, Δnos_i, $i = 1, 2, \ldots, no\_vars$, are randomly generated numbers such that $para_{\min}^i \le os_i + \Delta nos_i \le para_{\max}^i$. The first new offspring ($j = 1$) is obtained according to (14) with that only one b_i (i being randomly generated within the range) is allowed to be one and all the others are zeros. The second new offspring is obtained according to (14) with that some b_i randomly chosen are set to be one and others are zero. The third new offspring is obtained according to (14) with all $b_i = 1$. These three new offspring will then be evaluated using the fitness function of (4). A real number will be generated randomly and compared with a user-defined number $p_a \in [0 \quad 1]$. If the real number is smaller than p_a, the one with the largest fitness value among the three new offspring will replace the chromosome with the smallest fitness f_s in the population. If the real number is larger than p_a, the first offspring $\mathbf{nos}_1$ will replace the chromosome with the smallest fitness value f_s in the population if $f(\mathbf{nos}_1) > f_s$; the second and the third offspring will do the same. p_a is effectively the probability of accepting a bad offspring in order to reduce the chance of converging to a local optimum. Hence, the possibility of reaching the global optimum is kept.

In general, various methods like boundary mutation, uniform mutation, or nonuniform mutation [1], [32], [33] can be employed to realize the mutation operation. Boundary mutation is to change the value of a randomly selected gene to its upper or lower bound. Uniform mutation is to change the value of a randomly selected gene to a value between its upper and lower bounds. Nonuniform mutation is capable of fine-tuning the parameters by increasing or decreasing the value of a randomly selected gene by a weighted random number. The weight is usually a monotonic decreasing function of the number of iteration. In our approach, we have three offspring generated in the mutation process. From (14), the first mutation is in fact the uniform mutation. The second mutation allows some randomly selected genes to change simultaneously. The third mutation changes all genes simultaneously. The second and the third mutations allow multiple genes to be changed. Hence, the searching domain is larger than that formed by changing a single gene. The genes will have a larger space for improving when the fitness values are small. On the contrary, when the fitness values are nearly the same, changing the value of a single gene (the first mutation) will give a higher probability of improving the fitness value as the searching domain is smaller and some genes may have reached their optimal values.

After the operation of selection, crossover, and mutation, a new population is generated. This new population will repeat the same process. Such an iterative process can be terminated when the result reaches a defined condition, e.g., the change of the fitness values between the current and the previous iteration is less than 0.001, or a defined number of iteration has been reached.

III. Benchmark Test Functions

Some benchmark test functions [3], [4], [6], [17] are used to examine the applicability and efficiency of the improved GA. Six test functions $f_i(\mathbf{x}), i = 1, 2, 3, 4, 5, 6$ will be used, where $\mathbf{x} = [\, x_1 \quad x_2 \quad \cdots \quad x_n \,]^{\mathrm{T}}$. n is an integer denoting the dimension of the vector $\mathbf{x}$. The six test functions are defined as follows:

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2, \qquad -5.12 \le x_i \le 5.12 \tag{15}$$

where $n = 3$ and the minimum point is at $f_1(0, 0, 0) = 0$

$$\begin{aligned} f_2(\mathbf{x}) = \sum_{i=1}^{n-1} \left(100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right) \\ -2.048 \le x_i \le 2.048 \end{aligned} \tag{16}$$

where $n = 2$ and the minimum point is at $f_2(0, 0) = 0$

$$f_3(\mathbf{x}) = 6n + \sum_{i=1}^{n} floor(x_i), \quad -5.12 \le x_i \le 5.12 \tag{17}$$

where $n = 5$ and the minimum point is at $f_3([5.12, 5], \ldots, [5.12, 5]) = 0$. The floor function, $floor(\cdot)$, is to round down the argument to an integer

$$f_4(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + Gauss(0, 1), \quad -1.28 \le x_i \le 1.28 \tag{18}$$

where $n = 3$ and the minimum point is at $f_4(0, 0, 0) = 0$. $Gauss(0, 1)$ is a function to generate uniformly a floating-point number between zero and one inclusively

$$f_5(\mathbf{x}) = \frac{1}{k} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}$$
$$-65.356 \le x_i \le 65.356 \tag{19}$$

where we have $\mathbf{a}$ as shown in the equation at the bottom of the next page, and $k = 500$ and the maximum point is at $f_5(-32, -32) \approx 1$

$$f_6(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right], \quad -5.12 \le x_i \le 5.12 \tag{20}$$

where $n = 3$ and the minimum point is at $f_6(0, 0, 0) = 0$. It should be noted that the minimum values of all functions in the defined domain are zero except for $f_5(\mathbf{x})$. The fitness functions for f_1 to f_4 and f_6 are defined as

$$fitness = \frac{1}{1 + f_i(\mathbf{x})}, \qquad i = 1, 2, 3, 4, 6 \tag{21}$$

and the fitness function for f_5 is defined as

$$fitness = f_5(\mathbf{x}). \tag{22}$$

and we have the equation shown at the bottom of the page.

The proposed GA goes through these six test functions. The results are compared with those obtained by the standard GA with arithmetic crossover and nonuniform mutation [1], [31]–[33]. For each test function, the simulation takes 500 iterations and the population size is ten for the proposed and the standard GAs. The probability of crossover is set at 0.8 for all functions and the probability of mutation for functions f_1 to f_6 are 0.8, 0.8, 0.7, 0.8, 0.8, and 0.35, respectively. The shape parameters b of the standard GA [2] for nonuniform mutation, which is selected by trial and error through experiments for good performance, are set at $b = 5$ for f_1, f_2 and f_5, $b = 0.1$ for f_3, $b = 1$ for f_4 and f_6. For the proposed GA, the values of w are set to be 0.5, 0.99, 0.1, 0.5, 0.01, and 0.01 for the six test functions, respectively. The probability of acceptance p_a is set at 0.1 for all functions. These values are selected by trial and error through experiments for good performance. The initial values of $\mathbf{x}$ in the population for a test function are set to be the same for both the proposed and the standard GAs. For tests 1 to 6, the initial values are $[1 \quad 1 \quad 1]$, $[0.5 \quad 0.5]$, $[1 \quad \cdots \quad 1]$, $[0.5 \quad \cdots \quad 0.5]$, $[10 \quad \cdots \quad 10]$ and $[1 \quad 1 \quad 1]$, respectively. The results of the average fitness values over 100 times of simulations based on the proposed and standard GAs are shown in Fig. 3 and tabulated in Table I. Generally, it can be seen that the performance of the proposed GA is better than that of the standard GA.

IV. Neural Network With Link Switches and Tuning Using the Improved GA

In this section, a neural network with link switches is presented. By introducing a switch to a link, the parameters and the structure of the neural network can be tuned using the improved GA.

A. *Neural Network With Link Switches*

Neural networks [5] for tuning usually have a fixed structure. The number of connections may be too large for a given application such that the network structure is unnecessarily complex and the implementation cost is high. In this section, a multiple-input–multiple-output (MIMO) three-layer neural network is proposed as shown in Fig. 4. The main different point is that a unit step function is introduced to each link. Such a unit step function is defined as

$$\delta(\alpha) = \begin{cases} 0 & \text{if } \alpha < 0 \\ 1 & \text{if } \alpha \ge 0 \end{cases}, \quad \alpha \in \Re. \tag{23}$$

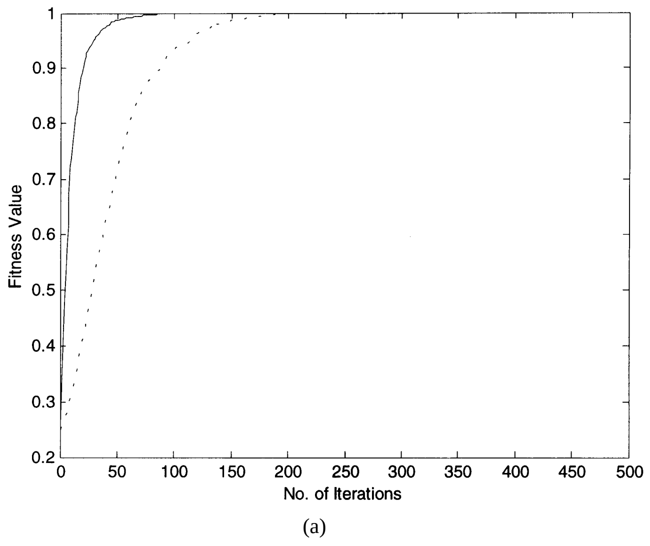
This is equivalent to adding a switch to each link of the neural network. Referring to Fig. 4, the input–output relationship of the proposed MIMO three-layer neural network is as follows:

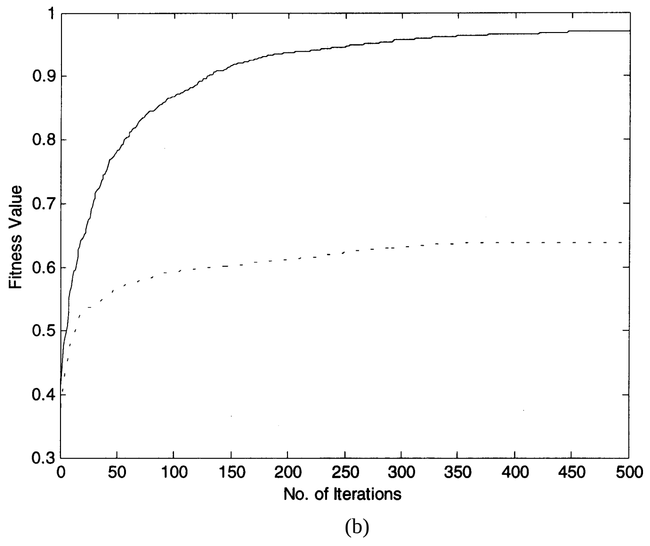
$$y_k(t) = \sum_{j=1}^{n_h} \delta\left(s_{jk}^2\right) w_{jk} \operatorname{logsig}\left[\sum_{i=1}^{n_{in}} \left(\delta(s_{ij}^1) v_{ij} z_i(t) - \delta(s_j^1) b_j^1\right)\right] - \delta\left(s_k^2\right) \operatorname{logsig}(b_k^2), \qquad k = 1, 2, \ldots, n_{out}. \tag{24}$$

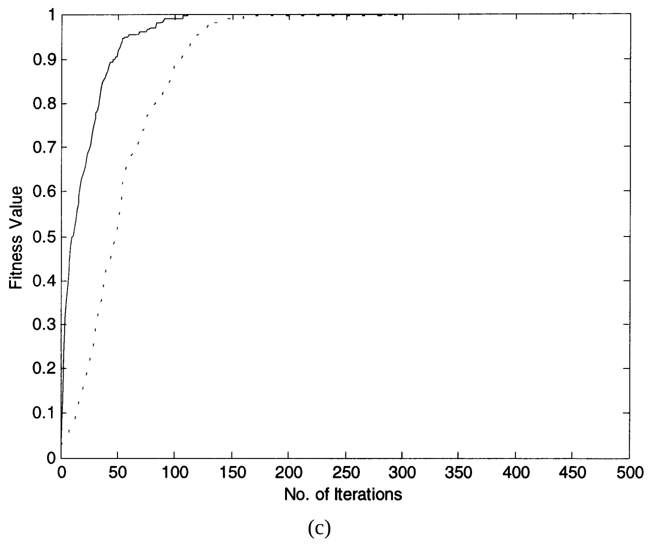
$z_i(t)$, $i = 1, 2, \ldots, n_{in}$, are the inputs which are functions of a variable t; n_{in} denotes the number of inputs; n_h denotes the number of the hidden nodes; w_{jk}, $j = 1, 2, \ldots, n_h$; $k = 1, 2, \ldots, n_{out}$, denotes the weight of the link between the jth hidden node and the kth output; v_{ij} denotes the weight of the link between the ith input and the jth hidden node; s_{ij}^1 denotes the parameter of the link switch from the ith input to the jth hidden node; s_{jk}^2 denotes the parameter of the link switch from the jth hidden node to the kth output; n_{out} denotes the number of outputs of the proposed neural network; b_j^1 and b_k^2 denote the biases for the hidden nodes and output nodes, respectively; s_j^1 and s_k^2 denote the parameters of the link switches of the biases to the hidden and output layers, respectively; $\operatorname{logsig}(\cdot)$ denotes the logarithmic sigmoid function:

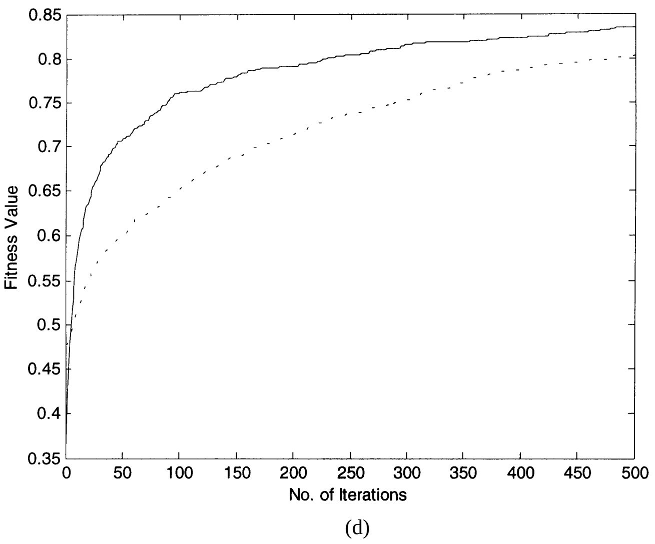
$$\operatorname{logsig}(\alpha) = \frac{1}{1 + e^{-\alpha}}, \qquad \alpha \in \Re. \tag{25}$$

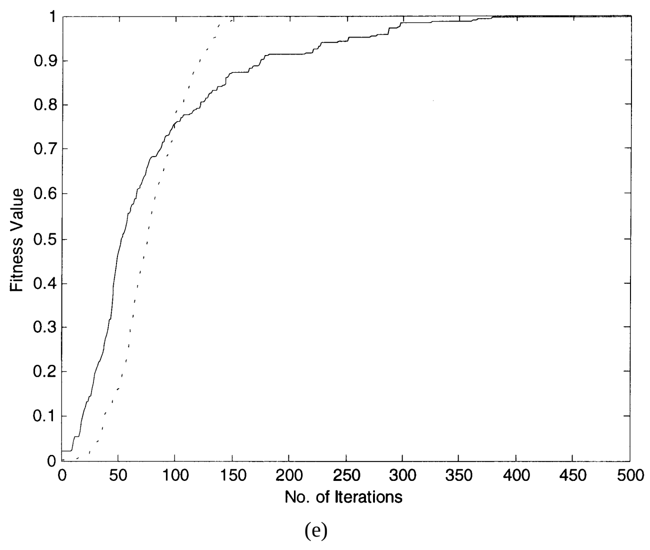
$$\mathbf{a} = \{a_{ij}\} = \left[\begin{array}{ccccccccccccccccccccccccc} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 \\ 32 & 32 & 32 & 32 & 32 & -16 & -16 & -16 & -16 & -16 & 0 & 0 & 0 & 0 & 0 & 16 & 16 & 16 & 16 & 16 & 32 & 32 & 32 & 32 & 32 \end{array}\right].$$











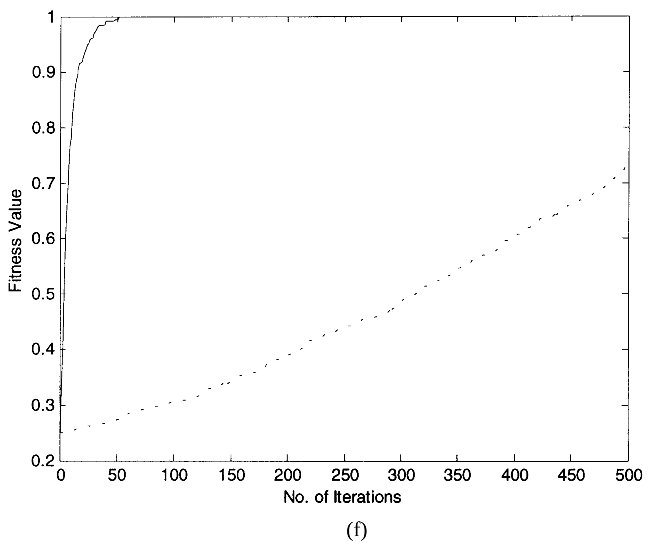


Fig. 3. Simulation results of the improved and standard GAs. The averaged fitness value of test functions obtained by the improved (solid line) and standard (dotted line) GAs. (a) $f_1(\mathbf{x})$. (b) $f_2(\mathbf{x})$. (c) $f_3(\mathbf{x})$. (d) $f_4(\mathbf{x})$. (e) $f_5(\mathbf{x})$. (f) $f_6(\mathbf{x})$.

$y_k(t)$, $k = 1, 2, \ldots, n_{out}$, is the kth output of the proposed neural network. By introducing the switches, the weights w_{jk} and v_{ij}, and the switch states can be tuned. It can be seen that the weights of the links govern the input–output relationship of the neural network while the switches of the links govern the structure of the neural network.

TABLE I
SIMULATION RESULTS OF THE PROPOSED GA AND THE STANDARD GA BASED ON THE BENCHMARK TEST FUNCTIONS

Test function	Proposed GA	Standard GA
$f_1(\mathbf{x})$	1.0000	1.0000
$f_2(\mathbf{x})$	0.9707	0.6393
$f_3(\mathbf{x})$	1.0000	1.0000
$f_4(\mathbf{x})$	0.8349	0.8037
$f_5(\mathbf{x})$	1.0000	1.0000
$f_6(\mathbf{x})$	1.0000	0.7297

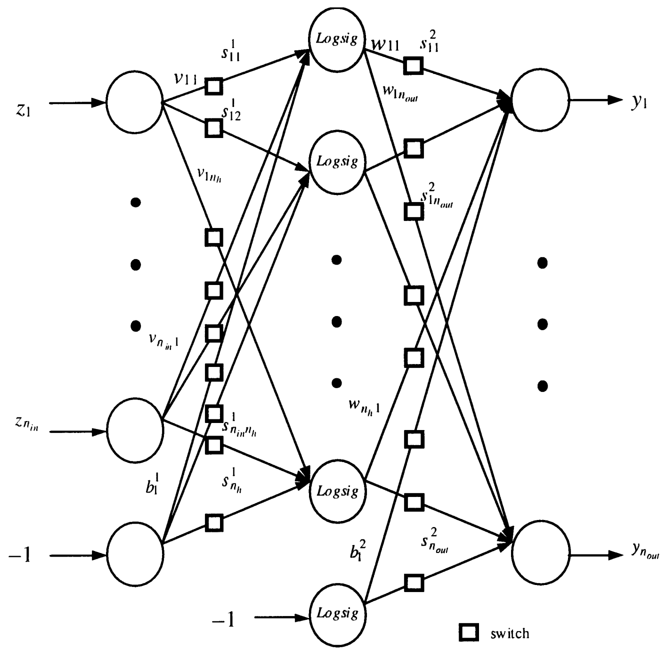


Fig. 4. Proposed three-layer neural network.

B. Tuning of the Parameters and Structure

The proposed neural network can be employed to learn the input–output relationship of an application by using the improved GA. The input–output relationship is described by

$$\mathbf{y}^d(t) = \mathbf{g}\left(\mathbf{z}^d(t)\right), \qquad t = 1, 2, \ldots, n_d \tag{26}$$

where $\mathbf{z}^d(t) = [z_1^d(t) \quad z_2^d(t) \quad \cdots \quad z_{n_{in}}^d(t)]$ and $\mathbf{y}^d(t) = [y_1^d(t) \ y_2^d(t) \ \cdots \ y_{n_{out}}^d(t)]$ are the given inputs and the desired outputs of an unknown nonlinear function $\mathbf{g}(\cdot)$ respectively, n_d denotes the number of input–output data pairs. The fitness function is defined as

$$\text{fitness} = \frac{1}{1+\text{err}} \tag{27}$$

$$\text{err} = \sum_{k=1}^{n_{out}} \frac{\sum_{t=1}^{n_d} \left|y_k^d(t) - y_k(t)\right|}{n_d}. \tag{28}$$

The objective is to maximize the fitness value of (27) using the improved GA by setting the chromosome to be $[s_{jk}^2 \ w_{jk} \ s_{ij}^1 \ v_{ij} \ s_j^1 \ b_j^1 \ s_k^2 \ b_k^2]$ for all i, j, k. It can be seen from (27) and (28) that a larger fitness value implies a smaller error value.

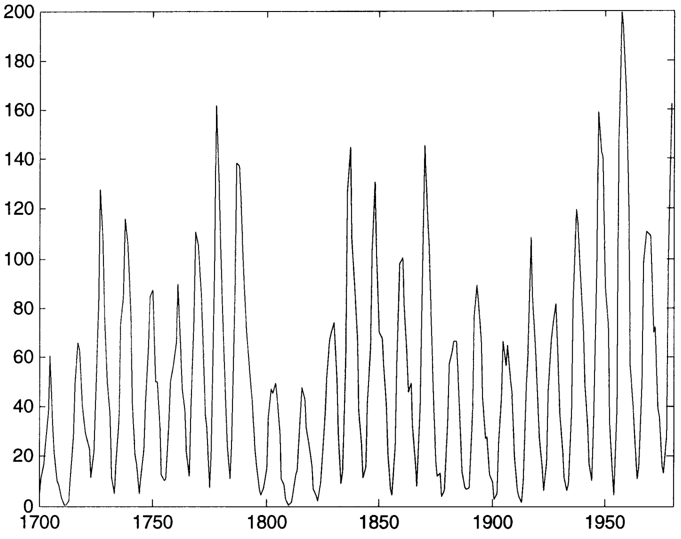


Fig. 5. Sunspot cycles from year 1700 to 1980.

V. APPLICATION EXAMPLES

Two application examples will be given in this section to illustrate the merits of the proposed neural networks tuned by the improved GA.

A. Forecasting of the Sunspot Number

An application example on forecasting the sunspot number [7], [8], [27] will be given in this section. The sunspot cycles from 1700 to 1980 are shown in Fig. 5. The cycles generated are nonlinear, nonstationary, and non-Gaussian which are difficult to model and predict. We use the proposed three-layer neural network (three-input-single-output) with link switches for the sunspot number forecasting. The inputs, z_i, of the purposed neural network are defined as $z_1(t) = y_1^d(t-1)$, $z_2(t) = y_1^d(t-2)$, and $z_3(t) = y_1^d(t-3)$, where t denotes the year and $y_1^d(t)$ is the sunspot numbers at the year t. The sunspot numbers of the first 180 years (i.e., $1705 \leq t \leq 1884$) are used to train the proposed neural network. Referring to (24), the proposed neural network used for the sunspot forecasting is governed by

$$y_1(t) = \sum_{j=1}^{n_h} \delta\left(s_{j1}^2\right) w_{j1} \cdot \operatorname{logsig}\left[\sum_{i=1}^{3} \left(\delta(s_{ij}^1) v_{ij} z_i(t) - \delta(s_j^1) b_j^1\right)\right] - \delta(s_1^2) \operatorname{logsig}(b_k^2. \tag{29}$$

The number of hidden nodes n_h is changed from three to seven to test the learning performance. The fitness function is defined as follows:

$$\text{fitness} = \frac{1}{1+\text{err}} \tag{30}$$

$$\text{err} = \sum_{t=1705}^{1884} \frac{\left|y_1^d(t) - y_1(t)\right|}{180}. \tag{31}$$

The improved GA is employed to tune the parameters and structure of the neural network of (29). The objective is to maximize the fitness function of (30). The best fitness value is one and the worst one is zero. The population size used for the improved GA is ten, $w = 0.9$ and $p_a = 0.1$ for all values of

TABLE II
SIMULATION RESULTS FOR THE APPLICATION EXAMPLE OF FORECASTING THE SUNSPOT NUMBER AFTER 1000 ITERATIONS OF LEARNING

	Our Approach		Standard GA with proposed neural network	
n_h	Fitness Values	Number of Links	Fitness Values	Number of Links
4	0.9429	9	0.9357	9
5	0.9448	17	0.9348	9
6	0.9453	18	0.9385	11
7	0.9426	13	0.9402	14
8	0.9407	13	0.9261	5

	Standard GA with traditional neural network		Back-Propagation with traditional neural network	
n_h	Fitness Values	Number of Links	Fitness Values	Number of Links
4	0.9356	21	0.9242	21
5	0.9306	26	0.9034	26
6	0.9401	31	0.8961	31
7	0.9402	36	0.8919	36
8	0.9366	41	0.8919	41

TABLE III
TRAINING ERROR AND FORECASTING ERROR IN MEAN ABSOLUTE ERROR (MAE) FOR THE APPLICATION EXAMPLE OF FORECASTING THE SUNSPOT NUMBER

	Our Approach		Standard GA with proposed neural network	
n_h	Training error	Forecasting error	Training error	Forecasting error
4	12.1116	13.9734	13.7473	14.6301
5	11.6850	13.8354	13.9495	15.7997
6	11.5730	14.0933	13.1060	14.8927
7	12.1791	14.7434	12.7207	13.9798
8	12.6076	14.3516	15.9594	19.5594

	Standard GA with traditional neural network		Back-Propagation with traditional neural network	
n_h	Training error	Forecasting error	Training error	Forecasting error
4	13.7666	15.6682	16.4123	20.1037
5	14.9151	15.7705	21.3844	26.2723
6	12.7433	16.9341	23.1991	28.5170
7	12.7207	14.1280	24.2294	29.6302
8	13.5383	14.2857	24.2323	29.8156

n_h. The lower and the upper bounds of the link weights are defined as $-3/\sqrt{n_h+1} \geq v_{ij}, w_{jk}, b_j^1, b_1^2 \geq 3/\sqrt{n_h+1}$ and, $-1 \geq s_{j1}^2, s_{ij}^1, s_j^1, s_1^2 \geq 1$, $i = 1, 2, \ldots, 3$; $j = 1, 2, \ldots, n_h$, $k = 1$ [16]. The chromosomes used for the improved GA are $[s_{j1}^2 \;\; w_{jk} \;\; s_{ij}^1 \;\; v_{ij} \;\; s_j^1 \;\; b_j^1 \;\; s_k^2 \;\; b_1^2]$. The initial values of all the link weights between the input and hidden layers are one, and those between the hidden and output layers are -1. The initial value of α in (23) is 0.5.

For comparison purpose, a fully connected three-layer feedforward neural network (three-input–one-output) [2] is also trained by the standard GA with arithmetic crossover and nonuniform mutation [1], [2], [5], and backpropagation with momentum and adaptive learning rate [30]. Also, the proposed neural network trained by the standard GA will be considered. For the standard GA, the population size is ten, the probability of crossover is 0.8 and the probability of mutation is 0.1. The shape parameters b of the standard GA with arithmetic crossover and nonuniform mutation, which is selected by trial and error through experiment for good performance, is set to be one. For the backpropagation with momentum and adaptive learning rate, the learning rate is 0.2, the ratio to increase the learning rate is 1.05, the ratio to decrease the learning rate is 0.7, the maximum validation failures is five, the maximum performance increase is 1.04, the momentum constant is 0.9. The initial values of the link weights are the same as those of the proposed neural network. For all approaches, the learning processes are carried out by a personal computer with a P4 1.4 GHz CPU. The number of iterations for all approaches is 1000.

The tuned neural networks are used to forecast the sunspot number during the years 1885–1980. The number of hidden nodes n_h is changed from four to eight. The simulation results for the comparisons are tabulated in Tables II and III. From Table II, it is observed that the proposed neural network trained with the improved GA provides better results in terms of accuracy (fitness values) and number of links. The training error [governed by (31)] and the forecasting error [governed by $\sum_{t=1885}^{1980} (|y_1^d(t) - y_1(t)|/96)$] are tabulated in Table III. Referring to Table III, the best result is obtained when the number of hidden node is six. Fig. 6 shows the simulation results of the forecasting using the proposed neural network trained with the improved GA (dashed lines) and the actual sunspot numbers (solid lines) for $n_h = 6$. The number of connected link is 18 after learning (the number of links of a fully connected network is 31 which includes the bias links). It is about 42% reduction of links. The training error and the forecasting error are 11.5730 and 14.0933, respectively.

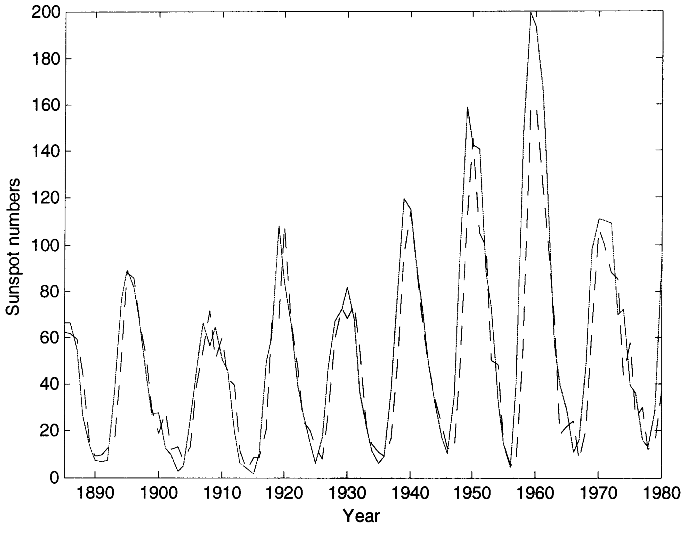


Fig. 6. Simulation results of a 96-year prediction using the proposed neural network ($n_h = 6$) with the proposed GA (dashed line), and the actual sunspot numbers (solid line) for the years 1885–1980.

B. Associative Memory

Another application example on tuning an associative memory will be given in this section. In this example, the associative memory, which maps its input vector into itself, has ten inputs and ten outputs. Thus, the desired output vector is its input vector. Referring to (24), the proposed neural network is given by

$$y_k(t) = \sum_{j=1}^{n_h} \delta\left(s_{jk}^2\right) w_{jk} \cdot \log\text{sig}\left[\sum_{i=1}^{10} \left(\delta(s_{ij}^1) v_{ij} z_i(t) - \delta(s_j^1) b_j^1\right)\right] - \delta(s_k^2) \log\text{sig}(b_k^2)$$
$$i = 1, 2, \ldots, 10, \quad k = 1, 2, \ldots, 10. \quad (32)$$

TABLE IV
SIMULATION RESULTS FOR THE APPLICATION EXAMPLE OF ASSOCIATIVE MEMORY AFTER 500 ITERATIONS OF LEARNING

	Our Approach		Standard GA with proposed neural network	
n_h	Fitness Values	Number of Links	Fitness Values	Number of Links
4	0.9648	79	0.9660	87
5	0.9629	91	0.9682	110
6	0.9651	113	0.9688	132
7	0.9642	137	0.9647	149
8	0.9607	153	0.9649	173

	Standard GA with traditional neural network		Back-Propagation with traditional neural network	
n_h	Fitness Values	Number of Links	Fitness Values	Number of Links
4	0.9678	94	0.9678	94
5	0.9681	115	0.9669	115
6	0.9668	136	0.9677	136
7	0.9688	157	0.9677	157
8	0.9694	178	0.9677	178

50 input vectors (each input vector has the property that $\mathbf{Z}9t) = 1$ to test the learning performance. The fitness function is defined as follows:

$$\text{fitness} = \frac{1}{1 + \text{err}} \tag{33}$$

$$\text{err} = \sum_{k=1}^{10} \frac{\sum_{t=1}^{50} |z_k(t) - y_k(t)|}{500}. \tag{34}$$

The improved GA is employed to tune the parameters and structure of the neural network of (32). The objective is to maximize the fitness function of (33). The best fitness value is one and the worst one is zero. The population size used for the improved GA is ten; $w = 0.8$ and $p_a = 0.1$ for all values of n_h. The lower and the upper bounds of the link weights are defined as $-3/\sqrt{n_h+1} \geq v_{ij}, w_{jk}, b_j^1, b_k^2 \geq 3/\sqrt{n_h+1}$ and $-1 \geq s_{jk}^2, s_{ij}^1, s_j^1, s_k^2 \geq 1$, $i = 1, 2, 3$; $j = 1, 2, \ldots, n_h$, $k = 10$ [16]. The chromosomes used for the improved GA are $[s_{jk}^2 \quad w_{jk} \quad s_{ij}^1 \quad v_{ij} \quad s_j^1 \quad b_j^1 \quad s_k^2 \quad b_k^2]$. The initial values of the link weights are all zero. For comparison purpose, a fully connected three-layer feedforward neural network (ten-input–ten-output) [2] trained by the standard GA (with arithmetic crossover and nonuniform mutation) and another trained by backpropagation (with momentum and adaptive learning rate) are considered again. Also, the proposed neural network trained by the standard GA will be considered. For the standard GA, the population size is ten, the probability of crossover is 0.8 and the probability of mutation is 0.03. The shape parameters b of the standard GA with arithmetic crossover and nonuniform mutation, which is selected by trial and error through experiments for good performance, is set to be three. For the backpropagation with momentum and adaptive learning rate, the learning rate is 0.2, the ratio to increase the learning rate is 1.05, the ratio to decrease the learning rate is 0.7, the maximum validation failures is five, the maximum performance increase is 1.04, the momentum constant is 0.9. The initial values of the links weights are the same as those of the proposed approach. The number of iterations for all approaches is 500. The simulation results are tabulated in Table IV. It can be seen from Table IV that the fitness values for different approaches are similar, but our approach can offer a smaller network.

VI. CONCLUSION

An improved GA has been proposed in this paper. By using the benchmark test functions, it has been shown that the improved GA performs more efficiently than the standard GA. Besides, by introducing a switch to each link, a neural network that facilitates the tuning of its structure has been proposed. Using the improved GA, the proposed neural network is able to learn both the input–output relationship of an application and the network structure. As a result, a given fully connected neural network can be reduced to a partially connected network after learning. This implies that the cost of implementation of the neural network can be reduced. Application examples on forecasting the sunspot number and tuning an associative memory using the proposed neural network trained with the improved GA have been given. The simulation results have been compared with those obtained by the proposed network trained by the standard GA, and traditional feedforward networks trained by the standard GA (with arithmetic crossover and nonuniform mutation) and the backpropagation (with momentum and adaptive learning rate).

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.

[2] D. T. Pham and D. Karaboga, *Intelligent Optimization Techniques, Genetic Algorithms, Tabu Search, Simulated Annealing and Neural Networks*. New York: Springer-Verlag, 2000.

[3] Y. Hanaki, T. Hashiyama, and S. Okuma, "Accelerated evolutionary computation using fitness estimation," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, vol. 1, 1999, pp. 643–648.

[4] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, Univ. Michigan, Ann Arbor, MI, 1975.

[5] Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolution Programs*, 2nd extended ed. New York: Springer-Verlag, 1994.

[6] G. X. Yao and Y. Liu, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, pp. 82–102, July 1999.

[7] M. Li, K. Mechrotra, C. Mohan, and S. Ranka, "Sunspot numbers forecasting using neural network," in *Proc. 5th IEEE Int. Symp. Intell. Contr.*, 1990, pp. 524–528.

[8] T. J. Cholewo and J. M. Zurada, "Sequential network construction for time series prediction," in *Proc. Int. Conf. Neural Networks*, vol. 4, 1997, pp. 2034–2038.

[9] B. D. Liu, C. Y. Chen, and J. Y. Tsao, "Design of adaptive fuzzy logic controller based on linguistic-hedge concepts and genetic algorithms," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 32–53, Feb. 2001.

[10] Y. S. Zhou and L. Y. Lai, "Optimal design for fuzzy controllers by genetic algorithms," *IEEE Trans. Ind. Applicat.*, vol. 36, pp. 93–97, Jan. 2000.

[11] C. F. Juang, J. Y. Lin, and C. T. Lin, "Genetic reinforcement learning through symbiotic evolution for fuzzy controller design," *IEEE Trans. Syst., Man. Cybern. B*, vol. 30, pp. 290–302, Apr. 2000.

[12] H. Juidette and H. Youlal, "Fuzzy dynamic path planning using genetic algorithms," *Electron. Lett.*, vol. 36, no. 4, pp. 374–376, Feb. 2000.

[13] R. Caponetto, L. Fortuna, G. Nunnari, L. Occhipinti, and M. G. Xibilia, "Soft computing for greenhouse climate control," *IEEE Trans. Fuzzy Syst.*, vol. 8, pp. 753–760, Dec. 2000.

[14] M. Setnes and H. Roubos, "GA-fuzzy modeling and classification: Complexity and performance," *IEEE Trans. Fuzzy Syst.*, vol. 8, pp. 509–522, Oct. 2000.

[15] K. Belarbi and F. Titel, "Genetic algorithm for the design of a class of fuzzy controllers: An alternative approach," *IEEE Trans. Fuzzy Systems*, vol. 8, pp. 398–405, Aug. 2000.

[16] M. Brown and C. Harris, *Neural Fuzzy Adaptive Modeling and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1994.

[17] S. Amin and J. L. Fernandez-Villacanas, "Dynamic local search," in *Proc. 2nd Int. Conf. Genetic Algorithms in Engineering Systems: Innovations and Applications*, 1997, pp. 129–132.

[18] X. Yao, "Evolving artificial networks," *Proc. IEEE*, vol. 87, pp. 1423–1447, July 1999.

[19] X. Yao and Y. Liu, “A new evolutionary system for evolving artificial neural networks,” *IEEE Trans. Neural Networks*, vol. 8, pp. 694–713, May 1997.

[20] F. J. Lin, C. H. Lin, and P. H. Shen, “Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive,” *IEEE Trans. Fuzzy Syst.*, vol. 9, pp. 751–759, Oct. 2001.

[21] Y. Hirose, K. Yamashita, and S. Hijiya, “Back-propagation algorithm which varies the number of hidden units,” *Neural Networks*, vol. 4, no. 1, pp. 61–66, 1991.

[22] A. Roy, L. S. Kim, and S. Mukhopadhyay, “A polynomial time algorithm for the construction and training of a class of multiplayer perceptions,” *Neural Networks*, vol. 6, no. 4, pp. 535–545, 1993.

[23] N. K. Treadold and T. D. Gedeon, “Exploring constructive cascade networks,” *IEEE Trans. Neural Networks*, vol. 10, pp. 1335–1350, Nov. 1999.

[24] C. C. Teng and B. W. Wah, “Automated learning for reducing the configuration of a feedforward neural network,” *IEEE Trans. Neural Networks*, vol. 7, pp. 1072–1085, Sept. 1996.

[25] Y. Q. Chen, D. W. Thomás, and M. S. Nixon, “Generating-shrinking algorithm for learning arbitrary classification,” *Neural Networks*, vol. 7, no. 9, pp. 1477–1489, 1994.

[26] M. C. Moze and P. Smolensky, “Using relevance to reduce network size automatically,” *Connect. Sci.*, vol. 1, no. 1, pp. 3–16, 1989.

[27] H. K. Lam, S. H. Ling, F. H. F. Leung, and P. K. S. Tam, “Tuning of the structure and parameters of neural network using an improved genetic algorithm,” in *Proc. 27th Annu. Conf. IEEE Ind. Electron. Soc.*, Denver, CO, Nov. 2001, pp. 25–30.

[28] G. P. Miller, P. M. Todd, and S. U. Hegde, “Designing neural networks using genetic algorithms,” in *Proc. 3rd Int. Conf. Genetic Algorithms Applications*, 1989, pp. 379–384.

[29] N. Weymaere and J. Martens, “On the initialization and optimization of multiplayer perceptrons,” *IEEE Trans. Neural Networks*, vol. 5, pp. 738–751, Sept. 1994.

[30] S. S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1999.

[31] X. Wang and M. Elbuluk, “Neural network control of induction machines using genetic algorithm training,” in *Conf. Record 31st IAS Annual Meeting*, vol. 3, 1996, pp. 1733–1740.

[32] L. Davis, *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.

[33] M. Srinivas and L. M. Patnaik, “Genetic algorithms: A survey,” *IEEE Computer*, vol. 27, pp. 17–27, June 1994.

[34] J. D. Schaffer, D. Whitley, and L. J. Eshelman, “Combinations of genetic algorithms and neural networks: A survey of the state of the art,” in *Proc. Int. Workshop Combinations Genetic Algorithms Neural Networks*, 1992, pp. 1–37.

[35] S. Bornholdt and D. Graudenz, “General asymmetric neural networks and structure design by genetic algorithms: A learning rule for temporal patterns,” in *Proc. Int. Conf. Syst., Man, Cybern.*, vol. 2, 1993, pp. 595–600.

[36] V. Maniezzo, “Genetic evolution of the topology and weight distribution of neural networks,” *IEEE Trans. Neural Networks*, vol. 5, pp. 39–53, Jan. 1994.

[37] P. J. Angeline, G. M. Saunders, and J. B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, Jan. 1994.

Frank H. F. Leung (M’92) was born in Hong Kong in 1964. He received the B.Eng. and Ph.D. degrees in electronic engineering from the Hong Kong Polytechnic University in 1988 and 1992, respectively.

He joined the Hong Kong Polytechnic University in 1992 and is now an Associate Professor in the Department of Electronic and Information Engineering. He has published more than 100 research papers on computational intelligence, control and power electronics. At present, he is actively involved in the research on Intelligent Multimedia Home and electronic Book.

Dr. Leung is a Reviewer for many international journals and had helped the organization of many international conferences. He is a Chartered Engineer and a Member of the Institution of Electrical Engineers (IEE).

H. K. Lam received the B.Eng. (Hons.) and Ph.D. degrees form the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong, in 1995 and 2000, respectively.

He is currently a Research Fellow in the Department of Electronic and Information Engineering at The Hong Kong Polytechnic University. His current research interests include intelligent control and systems, computational intelligence, and robust control.

S. H. Ling received the B.Eng. (Hons.) degree from the Department of Electrical Engineering, The Hong Kong Polytechnic University, Hong Kong, in 1999. He is currently a Research Student in the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University. His research interests include evolutionary computation, fuzzy logic, neural networks, and intelligent homes.

Peter K. S. Tam received the B.E., M.E., and Ph.D. degrees from the University of Newcastle, Newcastle, Australia, in 1971, 1973, and 1976, respectively, all in electrical engineering.

From 1967 to 1980, he held a number of industrial and academic positions in Australia. In 1980, he joined The Hong Kong Polytechnic University, Hong Kong, as a Senior Lecturer. He is currently an Associate Professor in the Department of Electronic and Information Engineering. He has participated in the organization of a number of symposiums and conferences. His research interests include signal processing, automatic control, fuzzy systems, and neural networks.