Tuning of the Structure and Parameters of Neural Networks using an Improved Genetic Algorithm

H.K. Lam, S.H. Ling, F.H.F. Leung and P.K.S. Tam
Centre for Multimedia Signal Processing
Department of Electronic and Information Engineering
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong.

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. The number of hidden nodes should be chosen manually starting from a small number. The number of hidden nodes should be increased if the learning performance in terms of fitness value is not acceptable. Using the improved GA, the structure and the parameters of the neural network can be tuned. Application examples on sunspot forecasting and associative memory are given to show the merits of the improved GA and the proposed neural network.**

I. Introduction

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 non-uniform mutation can be found [1, 31-33].

Neural network was proved to be a universal approximator [16]. A 3-layer feed-forward 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 back propagation algorithm [2]. In general, the learning steps of a neural network are as follows. First, a network structure is defined with fixed numbers of inputs, hidden nodes and outputs. Second, an algorithm is chosen to realize the learning process. It can be seen that 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. Genetic algorithms [27-28] were employed to obtain the solution. Pattern-classification approaches [29] can also be found to design the network structure. Some other methods have been proposed to learn both the network structure and connection weights. The evolution cycle of these methods can be summarized by three steps as follows. 1) Evaluate each individual, i.e., chromosome, according to a defined fitness function. 2) Select individual for reproduction and genetic operation. 3) Apply different kinds of genetic operations to the chromosomes to obtain the next generation. An ANNA ELEONORA algorithm was proposed [36]. New genetic operator and encoding procedures (binary) which allows the algorithm to obtain an opportune length of the coding string were introduced. Each gene consists of two parts, connectivity bits and the connection weight bits. The connectivity bits are to indicate the absence or present of a link. The connection weight bits are related to the value of the weight of a link. A GNARL algorithm was also proposed in [37]. At first, the population of the chromosomes representing the network structure will be generated randomly. The number of hidden nodes and connection links for each network is randomly chosen within some defined ranges. Three steps were proposed to generate an offspring: copying the parents, determining the mutations to be performed and mutating the copy. The severity of mutations measuring the performance of the network will be used to anneal the structural and parametric similarity between parent and offspring. The networks with high similarity will be mutated severely, on the contrary, the networks with low similarity will be mutated slightly. Mutation of the copy is separated into two classes, parametric mutations which alter the connection weights and structural mutations alter the number of hidden nodes and the presence of links in the network. An evolutionary system named EPNet can also be found for evolving the neural networks. Rank-based selection and five mutations were employed to modify the network structure and connection weights. The five mutations are hybrid training, node deletion, connection deletion, connection addition and node addition. Hybrid training, based on a modified back-propagation with adaptive learning rate and simulated annealing, is the mutation used to modify the connection weights. The other four mutations are used to grow and prune hidden node nodes and connections of the network. Some other algorithms can also be found to evolve the network structure and connection weights simultaneously.

In this paper, a three-layer neural network with switches introduced in some links is proposed to facilitate the tuning of the network structure. As a result, for a given fully connected feedforward neural network, it may no longer be a fully connected network after learning. This implies that the cost of implementing the proposed neural network, in terms of hardware implementation and processing time, can be reduced. The network structure and 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 feed-forward networks [2] trained by (i) the standard GA with arithmetic crossover and non-uniform mutation [1-2, 5] and (ii) the back-propagation with momentum and adaptive learning rate [30].

This paper is organized as follows. In session II, the improved genetic algorithm is presented. In session 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 session IV, the neural network with link switches, and tuning of its structure and parameters using the improved GA, will be presented. Application examples will be presented in session V. A conclusion will be drawn in session VI.

II. IMPROVED GENETIC ALGORITHM

Genetic algorithms (GAs) are powerful searching algorithms. 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.

$$P = \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_{pop\_size}\} \quad (1)$$

$$\mathbf{p}_i = \begin{bmatrix} p_{i_1} & p_{i_2} & \cdots & p_{i_j} & \cdots & p_{i_{no\_vars}} \end{bmatrix}, \; i = 1, 2, \ldots, pop\_size;\; j = 1, 2, \ldots, no\_vars \quad (2)$$

$$para_{\min}^{j} \le p_{i_j} \le para_{\max}^{j} \quad (3)$$

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; $para_{\min}^{j}$ and $para_{\max}^{j}$ are the minimum and maximum values of the parameter p_{i_j} for all *i*. It can be seen from (1) to (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,

$$fitness = f(\mathbf{p}_i) \quad (4)$$

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 $\mathbf{p}_i$:

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

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

$$\hat{q}_i = \sum_{j=1}^{i} q_j, \ i = 1, 2, \ldots, pop\_size \tag{6}$$

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 $\hat{q}_{i-1} < d \le \hat{q}_i$, i = 1, 2, …, *pop_size*, and $\hat{q}_0 = 0$. 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. The details of the crossover operation are as follows. First, four chromosomes will be generated according to the following mechanisms,

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

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

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

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

$$\mathbf{p}_{\max} = \begin{bmatrix} para_{\max}^1 & para_{\max}^2 & \cdots & para_{\max}^{no\_vars} \end{bmatrix} \tag{11}$$

$$\mathbf{p}_{\min} = \begin{bmatrix} para_{\min}^1 & para_{\min}^2 & \cdots & para_{\min}^{no\_vars} \end{bmatrix} \tag{12}$$

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,

$$\mathbf{os} \equiv \begin{bmatrix} os_1 & os_2 & \cdots & os_{no\_\mathrm{vars}} \end{bmatrix} = \mathbf{os}_c^{i_{os}} \tag{13}$$

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

The offspring generated by the crossover operation will undergo the mutation operation. 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 may not be better than that form our approach. As seen from (7) to (10), the offspring spreads over the domain: (7) and (10) will move the offspring near centre region of the concerned domain (as w in (10) approaches 1, $\mathbf{os}_c^4$ approaches $\frac{\mathbf{p}_1 + \mathbf{p}_2}{2}$), and (8) and (9) will move the offspring near the domain boundary (as w in (8) and (9) approaches 1, $\mathbf{os}_c^2$ and $\mathbf{os}_c^3$ approaches $\mathbf{p}_{\max}$ 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:

$$\mathbf{nos}_j = \left[os_1 \quad os_2 \quad \cdots \quad os_{no\_\text{vars}} \right] + \left[b_1 \Delta nos_1 \quad b_2 \Delta nos_2 \quad \cdots \quad b_{no\_\text{vars}} \Delta nos_{no\_\text{vars}} \right],\ j = 1, 2, 3 \qquad (14)$$

where b_i, i = 1, 2, …, *no_vars,* can only take the value of 0 or 1, Δnos_i, i = 1, 2, …, *no_vars,* are randomly generated numbers such that $para^i_{\min} \le os^j_i + \Delta nos_i \le para^i_{\max}$. 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 1 and all the others are zeros. The second new offspring is obtained according to (14) with that some randomly chosen b_i are set to be 1 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 f_l 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 will replace the chromosome with the smallest fitness value f_s in the population if $f_l > f_s$; 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 non-uniform 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. Non-uniform mutation is capable of fine-tuning the parameters. The value of a randomly selected gene will be increased or decreased by a weighted random number. The weight is usually a monotonic decreasing function of the number of iterations. In our approach, we have three offspring generated in the mutation process. From (14), the first mutation is in fact a 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 domain to be searched is larger as compared with a domain characterized by changing a single gene. As the initial values are generated randomly, the genes will have a larger space for improving the fitness value when the fitness value is small. On the contrary, when the fitness values are large and nearly steady, changing the value of a single gene (the first mutation) may be enough as some genes may have reached the 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 iterations is 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 $\mathbf{x} = [x_1 \quad x_1 \quad \cdots \quad x_n]$. n is an integer denoting the dimension of the vector $\mathbf{x}$. The six test functions are defined as follows,

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

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

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

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

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

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

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

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 0 and 1 inclusively.

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

where

$$\mathbf{a} = \{a_{ij}\} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 & -32 & -16 & 0 & 16 & 32 \\ 32 & 32 & 32 & 32 & 32 & -16 & -16 & -16 & -16 & -16 & 0 & 0 & 0 & 0 & 0 & 16 & 16 & 16 & 16 & 16 & 32 & 32 & 32 & 32 & 32 \end{bmatrix},$$

k = 500 and the maximum point is at $f_5(-32, -32) \approx 1$.

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

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,

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

and the fitness function for f_5 is defined as,

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

The proposed GA goes through these 6 test functions. The results are compared with those obtained by the standard GA with arithmetic crossover and non-uniform mutation [1, 31-33]. For each test function, the simulation takes 500 iterations and the population size is 10 for the proposed and the standard GAs. When the standard GA is used, 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, 0.35 respectively. The shape parameters b of the standard GA [2] for non-uniform 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 has to be large enough to fit a given application. This may cause the neural network structure to be unnecessarily complex and increase the implementation cost. In this section, a multiple-input-multiple-output 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,

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

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 multiple-input multiple-output three-layer neural network is as follows,

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

$z_i(t)$, $i = 1, 2, \ldots, n_{in}$, 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}, $j = 1, 2, \ldots, n_h$; $k = 1, 2, \ldots, n_{out}$, denote the weight of the link between the j-th hidden node and the k-th output; v_{ij} denotes the weight of the link between the i-th input and the j-th hidden node; s_{ij}^1 denotes the parameter of the link switch from the i-th input to the j-th hidden node; s_{jk}^2 denotes the parameter of the link switch from the j-th hidden node to the k-th 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; $logsig(\cdot)$ denotes the logarithmic sigmoid function:

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

$y_k(t)$, $k = 1, 2, \ldots, n_{out}$, is the k-th 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.

B. Tuning of the Parameters and Structure

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

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

where $\mathbf{z}^d(t) = \begin{bmatrix} z_1^d(t) & z_2^d(t) & \cdots & z_{n_{in}}^d(t) \end{bmatrix}$ and $\mathbf{y}^d(t) = \begin{bmatrix} y_1^d(t) & y_2^d(t) & \cdots & y_{n_{out}}^d(t) \end{bmatrix}$ 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,

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

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

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

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 of the sunspot number [7-8, 27] will be given in this section. The sunspot numbers from 1700 to 1980 are shown in Fig. 5. The cycles generated are non-linear, non-stationary, and non-Gaussian which are difficult to model and predict. We use the proposed 3-layer neural network (3-input-single-output) with link switches for the sunspot number forecasting. The inputs, z_i, of the proposed 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. $1705 \le t \le 1884$) are used to train the proposed neural network. Referring to (24), the proposed neural network used for the sunspot forecasting is governed by,

$$y_1(t) = \sum_{j=1}^{n_h} \delta(s_{j1}^2) w_{j1} logsig\left[\sum_{i=1}^{3}\left(\delta(s_{ij}^1) v_{ij} z_i(t) - \delta(s_j^1) b_j^1\right)\right] - \delta(s_1^2) b_1^2 \tag{29}$$

The value of n_h is changed from 3 to 7 to test the learning performance. The fitness function is defined as follows,

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

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

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 1 and the worst one is 0. The population size used for the improved GA is 10; w = 0.9 and p_a = 0.1 for all values of n_h. The lower and the upper bounds of the link weights are defined as $\frac{-3}{\sqrt{n_h}} \ge v_{ij}, w_{jk}, b_j^1, b_1^2 \ge \frac{3}{\sqrt{n_h}}$ and, $-1 \ge s_{j1}^2, s_{ij}^1, s_j^1, s_1^2 \ge 1$, i = 1, 2, …, 3; j = 1, 2, …, n_h, k = 1 [16]. The chromosomes used for the improved GA are $\begin{bmatrix} s_{j1}^2 & w_{jk} & s_{ij}^1 & v_{ij} & s_j^1 & b_j^1 & s_k^2 & b_1^2 \end{bmatrix}$. The initial values of all the link weights between the input and hidden layers are 1 and those between the hidden and output layers are –1 respectively. The initial values of the switches are all 0.5.

For comparison purpose, a fully connected 3-layer feed-forward neural network (3-input-1-output) [2] is trained by (i) the standard GA with arithmetic crossover and non-uniform mutation [1-2, 5], and (ii) back-propagation with momentum and adaptive learning rate [30]. On the other hand, the proposed neural network is also trained with the standard GA for comparsion. For standard GA, the population size is 10, 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 non-uniform mutation, which is selected by trial and error through experiments for good performance, is set to be 1. For the back-propagation with momentum and adaptive learning rate, the learning rate is 0.2, the ratio to increase learning rate is 1.05, the ratio to decrease the learning rate is 0.7, the maximum validation failures is 5, the maximum performance increase is 1.04, and the momentum constant is 0.9. The initial values of the link weights are the same as those in the proposed neural network. For all approaches, the learning processes are carried out by a personal computer with a P4 1.4GHz 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. 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). The number of hidden nodes n_h is changed from 4 to 8. The simulation results for the comparisons are tabulated in Table II and Table III. From Table II, it is observed that the proposed neural network trained with the improved GA provides better results than those of the proposed neural network with standard GA, the traditional feed-forward neural network trained with standard GA and back-propagation with momentum and adaptive learning rate in terms of accuracy (fitness values) and number of links. The training error (governed by (31)) and the forecasting error (governed by $\sum_{t=1885}^{1980} \frac{\left| y_1^d(t) - y_1(t) \right|}{96}$) are tabulated in Table III. It can be observed from Table III that our approach performs better than the traditional approaches. Refer to Table III, the best result is obtained when the number of hidden node is 6. The number of connected link is 18 after learning (the number of links of a fully connected network is 31, including the bias links). It is about 41.9% reduction of the number of links after learning. The training error and the forecasting error in term of mean absolute error (MAE) are 11.5730 and 14.0933 respectively.

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 10 inputs and 10 outputs. Thus, the desired output vector is its input vector. Referring to (24), the proposed neural network used for the sunspot forecasting is reused,

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

50 sets of input vector (each input vector has the property that $\|\mathbf{z}(t)\| = 1$) will be employed to train the proposed neural network. The value of n_h is changed from 4 to 8 to test the learning performance. The fitness function is defined as follows,

$$fitness = \frac{1}{1+err} \quad (33)$$

$$err = \sum_{k=1}^{10} \frac{\sum_{t=1}^{100} |z_k(t) - y_k(t)|}{100} \quad (34)$$

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). A larger value of the fitness function indicates a smaller value of *err* of (34). The best fitness value is 1 and the worst one is 0. The population size used for the improved GA is 10; 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 $\frac{-3}{\sqrt{n_h}} \geq v_{ij}, w_{jk}, b_j^1, b_k^2 \geq \frac{3}{\sqrt{n_h}}$ and, $-1 \geq s_{jk}^2, s_{ij}^1, s_j^1, s_k^2 \geq 1$, i = 1, 2, …, 3; j = 1, 2, …, n_h, k = 10 [16]. The chromosomes used for the improved GA are $\begin{bmatrix} s_{jk}^2 & w_{jk} & s_{ij}^1 & v_{ij} & s_j^1 & b_j^1 & s_k^2 & b_k^2 \end{bmatrix}$. The initial values of the link weights are all zero. For comparison purpose, the proposed neural network trained by the standard GA (with arithmetic crossover and non-uniform mutation [1-2, 5]), a fully connected 3-layer feed-forward neural networks (10-input-10-output) [2] trained by the standard GA and back-propagation (with momentum and adaptive learning rate [30]) are used again. For the standard GA, the population size is 10, 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 non-uniform mutation, which is selected by trial and error through experiments for good performance, is set to be 3. For the back-propagation 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 5, the maximum performance increase is 1.04, and 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 n_h by the standard GA (with arithmetic crossover and non-uniform mutation) and the back-propagation (with momentum and adaptive learning rate) are similar to those by our approach, which offer smaller networks.

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 partly connected network after learning. This implies a lower cost of implementation of the neural network. Application examples on forecasting the sunspot numbers and tuning of 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 a traditional feed-forward network trained by (i) the standard GA with arithmetic crossover and non-uniform mutation, and (ii) the back-propagation with momentum and adaptive learning rate.

Acknowledgement

The work described in this paper was substantially supported by a Research Grant of Centre for

Multimedia Signal Processing, The Hong Kong Polytechnic University (project number A432).

REFERENCES

[1] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[2] D.T. Pham and D. Karaboga, *Intelligent optimization techniques, genetic algorithms, tabu search, simulated annealing and neural networks*, Springer, 2000.

[3] Y. Hanaki, T. Hashiyama and S. Okuma, “Accelerated evolutionary computation using fitness estimation,” *IEEE SMC '99 Conference Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 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. Thesis*, University of Michigan, Ann Arbor, MI., 1975.

[5] Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolution Programs*, second, extended edition, Springer-Verlag, 1994.

[6] G.X. Yao and Y. Liu "Evolutionary Programming made Faster," *IEEE Trans., on Evolutionary Computation*, vol. 3, no. 2, July 1999, pp.82-102

[7] M. Li, K. Mechrotra, C. Mohan and S. Ranka, “Sunspot Numbers Forecasting Using Neural Network,” *5th IEEE International Symposium on Intelligent Control, 1990. Proceedings*, pp.524-528, 1990.

[8] T.J. Cholewo, J.M. Zurada, “Sequential network construction for time series prediction,” *International Conference on Neural Networks*, vol.4, pp.2034-2038, 1997.

[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. Systems, Man and Cybernetics, Part B,* vol. 31 no. 1, Feb. 2001, pp. 32-53.

[10] Y.S. Zhou and L.Y. Lai “Optimal design for fuzzy controllers by genetic algorithms,” *IEEE Trans., Industry Applications,* vol. 36, no. 1, Jan.-Feb. 2000, pp. 93-97.

[11] C.F. Juang, J.Y. Lin and C.T. Lin, “Genetic reinforcement learning through symbiotic evolution for fuzzy controller design,” *IEEE Trans., Systems, Man and Cybernetics, Part B*, vol. 30, no. 2, April, 2000, pp. 290–302.

[12] H. Juidette and H. Youlal, “Fuzzy dynamic path planning using genetic algorithms,” *Electronics Letters*, vol. 36, no. 4, Feb. 2000, pp. 374-376.

[13] R. Caponetto, L. Fortuna, G. Nunnari, L. Occhipinti and M. G. Xibilia, “Soft computing for greenhouse climate control,” *IEEE Trans., Fuzzy Systems*, vol. 8, no. 6, Dec. 2000, pp. 753-760.

[14] M. Setnes and H. Roubos, “GA-fuzzy modeling and classification: complexity and performance,” *IEEE. Trans, Fuzzy Systems*, vol. 8, no. 5, Oct. 2000, pp. 509–522.

[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, no. 4, Aug. 2000, pp. 398-405.

[16] M. Brown and C. Harris, *Neuralfuzzy adaptive modeling and control*, Prentice Hall, 1994.

[17] S. Amin and J.L. Fernandez-Villacanas, “Dynamic Local Search,” *Second International Conference On Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp129-132, 1997.

[18] Xin Yao, “Evolving Artificial Networks,” *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1423-1447, 1999.

[19] Xin Yao and Young Liu, “A new Evolutionary System for Evolving Artificial Neural Networks,” *IEEE Tran. On Neural Networks*, vol. 8, no. 3, pp. 694-713, 1997.

[20] Faa-Jeng Lin, Chih-Hong Lin, Po-Hung Shen, “Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive,” *IEEE Trans., Fuzzy Systems*, vol. 9, no. 5, 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] Nicholas. K. Treadold and Tamas D. Gedeon, "Exploring constructive Cascade Networks," *IEEE Trans. Neural Networks*, vol. 10, no. 6, pp. 1335-1350, Nov, 1999.

[24] Chin-Chi Teng and Benjamin W. Wah, "Automated learning for reducing the configuration of a feedforward neural network," *IEEE Trans. Neural Networks*, vol. 7, no. 5, pp. 1072-1085, Sep, 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," *Proceedings of the 27th Annual Conference of the IEEE Industrial Electronics Society, IECON'2001*, Denver, Nov, 2001, pp. 25-30.

[28] G.P. Miller, P.M. Todd and S.U. Hegde, "Designing neural networks using genetic algorithms," Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 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] Haykin, Simon S., *Neural networks: A comprehensive foundation*, 2nd Ed., Upper Saddle River, N. J.: Prentice Hall, 1999.

[31] Xiufeng Wang and M., Elbuluk, "Neural network control of induction machines using genetic algorithm training," *Industry Applications Conference, 1996. Thirty-First IAS Annual Meeting, IAS '96., Conference Record of the 1996 IEEE*, vol. 3, 1996, pp. 1733-1740.

[32] Lawrence Davis, *Handbook of genetic algorithms*, Van Nostrand Reinhold, New York, 1991.

[33] M. Srinivas, L.M. Patnaik "Genetic algorithms: a survey," *IEEE Computer*, vol. 27, issue 6, pp.

17-26, 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,” *Proceedings of International Workshop on Combinations of Genetic Algorithms and Neural Networks, COGANN-92*, 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,” *Conference proceedings of International Conference on Systems, Man and Cybernetics*, 1993, 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, issue 1, pp. 39-53, Jan. 1994.

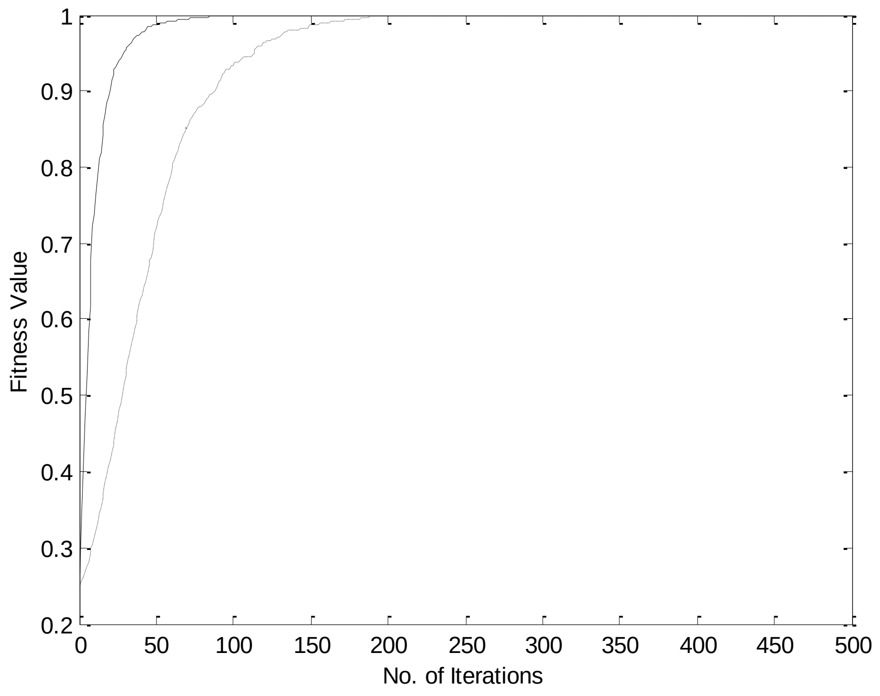
[37] P.J. Angeline, G.M. Saunders and J.B. Pollack, “An evolutionary algorithm that constructs recurrent neural networks,” *IEEE Trans. Neural Networks*, vol.5, issue 1, pp. 54-65, Jan. 1994

```
Procedure of the standard GA
begin
        $\tau \rightarrow 0$        // $\tau$: iteration generation
initialize $\mathbf{P}(\tau)$                    //$\mathbf{P}(\tau)$: population for iteration $t$
        evaluate $f(\mathbf{P}(\tau))$               // $f(\mathbf{P}(\tau))$:fitness function
while (not termination condition) do
    begin
        $\tau \rightarrow \tau+1$
        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
end
```

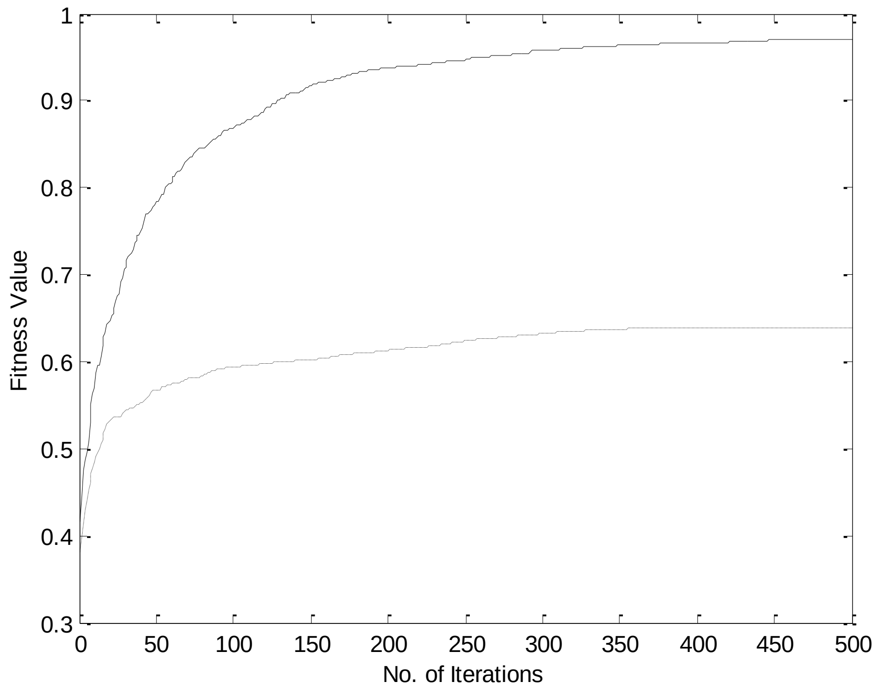
Fig. 1. Procedure of standard GA.

```
Procedure of the improved GA
begin
        $\tau \rightarrow 0$        // $\tau$: iteration
        initialize $\mathbf{P}(\tau)$                    //$\mathbf{P}(\tau)$: population for iteration $t$
        evaluate $f(\mathbf{P}(\tau))$      // $f(\mathbf{P}(\tau))$:fitness function
while (not termination condition) do
        begin
        $\tau \rightarrow \tau+1$
        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 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
                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
                -
```

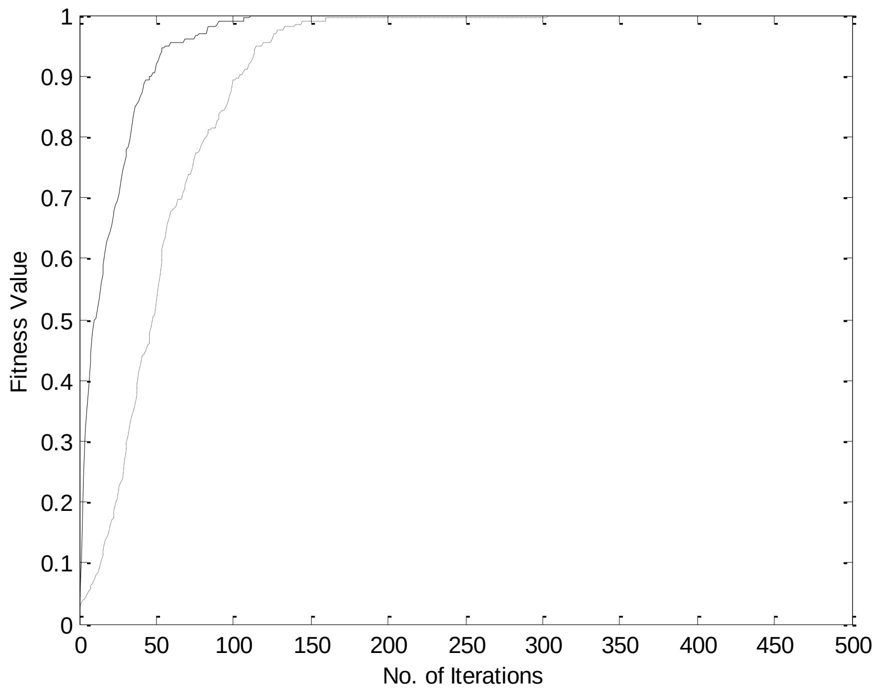
Fig. 2. Procedure of the improved GA.



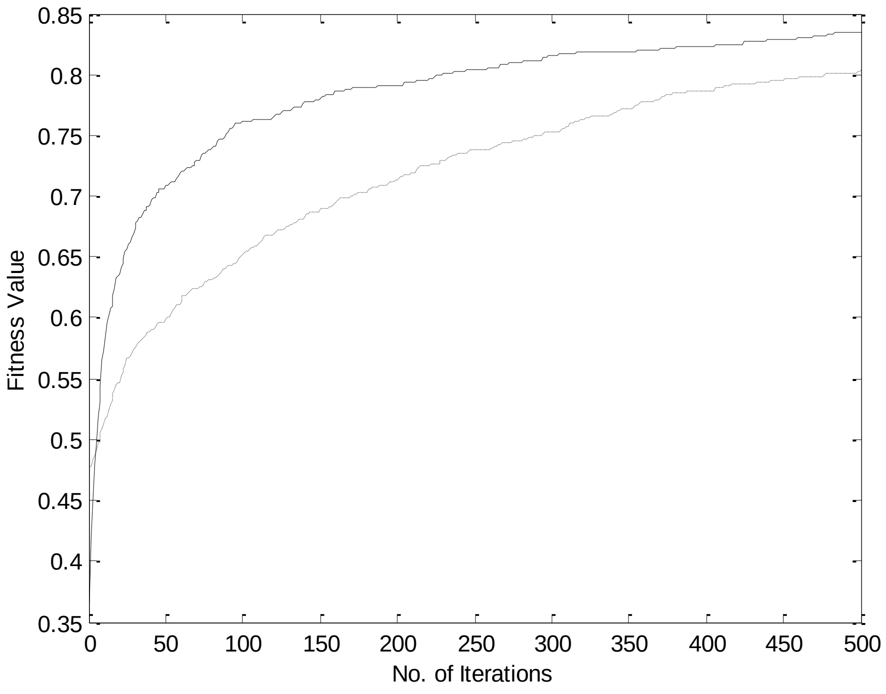
(a). The averaged fitness value of the test function $f_1(\mathbf{x})$ obtained by the improved (solid line) and standard (dotted line) GAs.



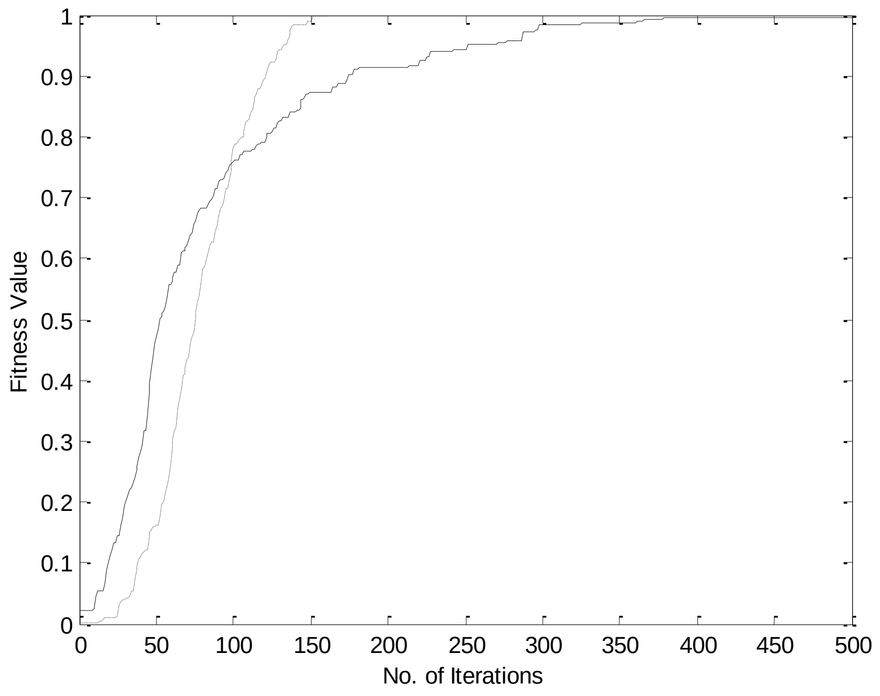
(b). The averaged fitness value of the test function $f_2(\mathbf{x})$ obtained by the improved (solid line) and standard (dotted line) GAs.



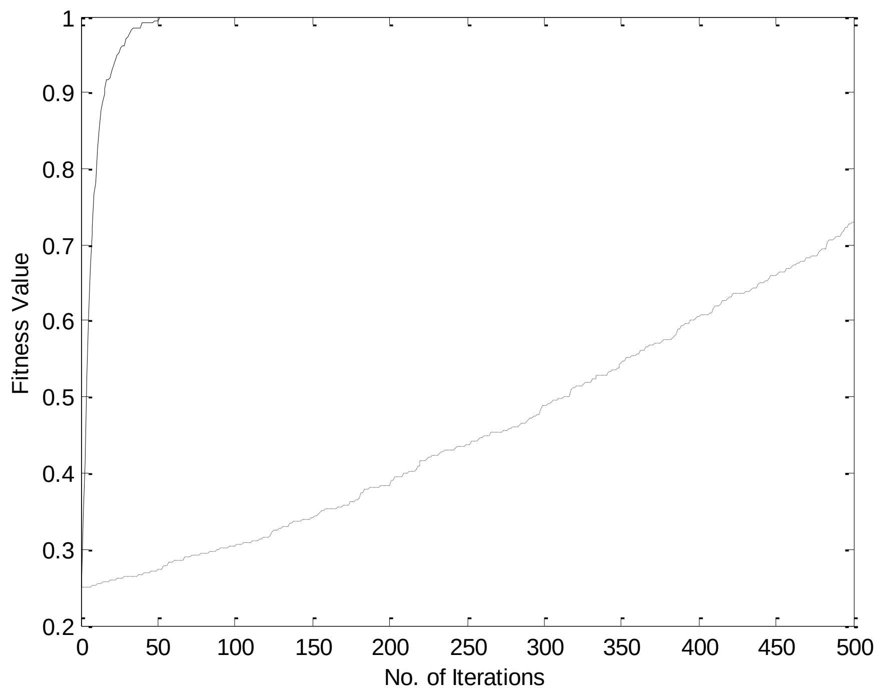
(c). The averaged fitness value of the test function $f_3(\mathbf{x})$ obtained by the improved (solid line) and standard (dotted line) GAs.



(d). The averaged fitness value of the test function $f_4(\mathbf{x})$ obtained by the improved (solid line) and standard (dotted line) GAs.



(e). The averaged fitness value of the test function $f_5(\mathbf{x})$ obtained by the improved (solid line) and standard (dotted line) GAs.



(f). The averaged fitness value of the test function $f_6(\mathbf{x})$ obtained by the improved (solid line) and standard (dotted line) GAs.

Fig. 3. Simulation results of the improved and standard GAs.

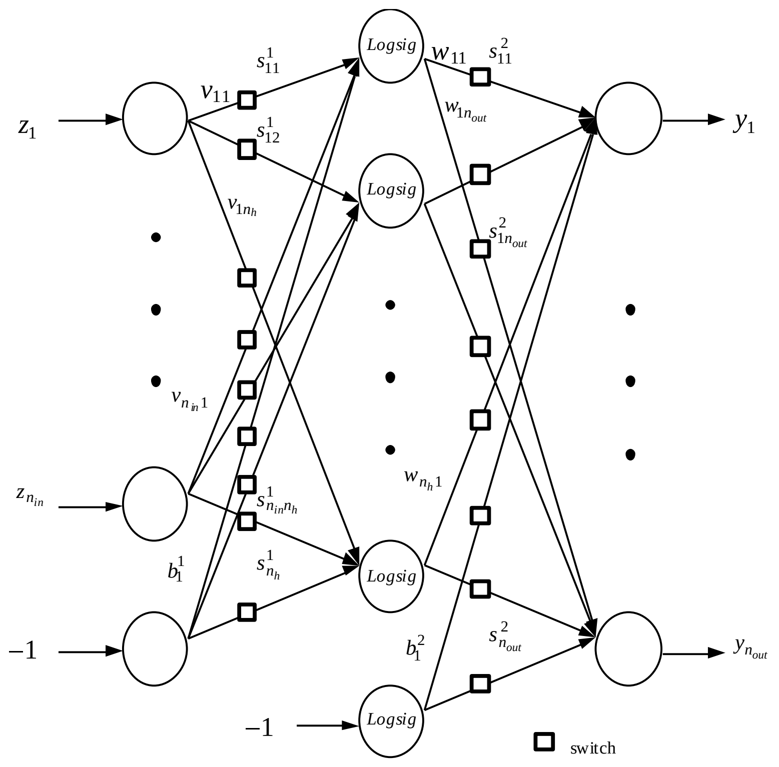


Fig. 4. Proposed 3-layer neural network.

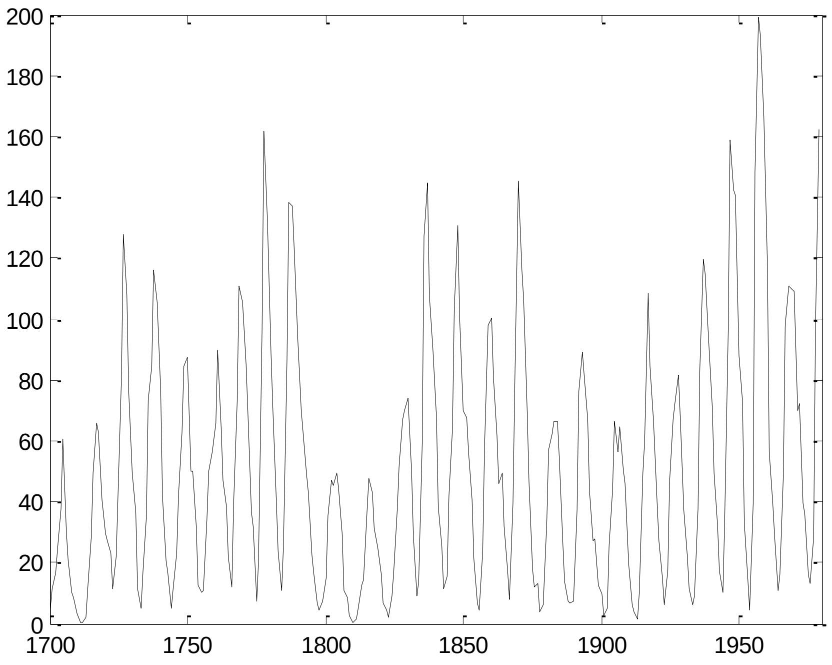
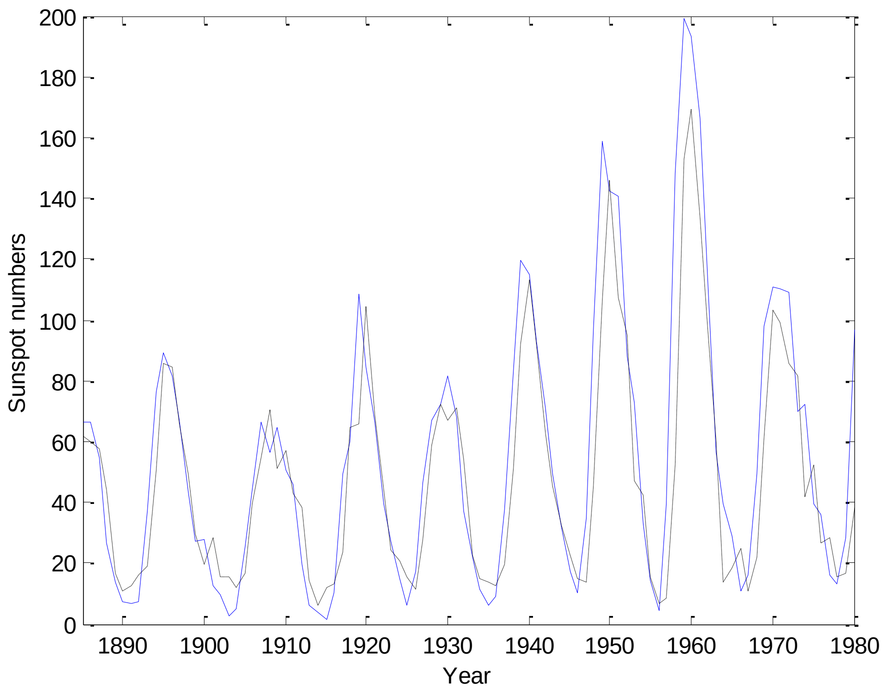
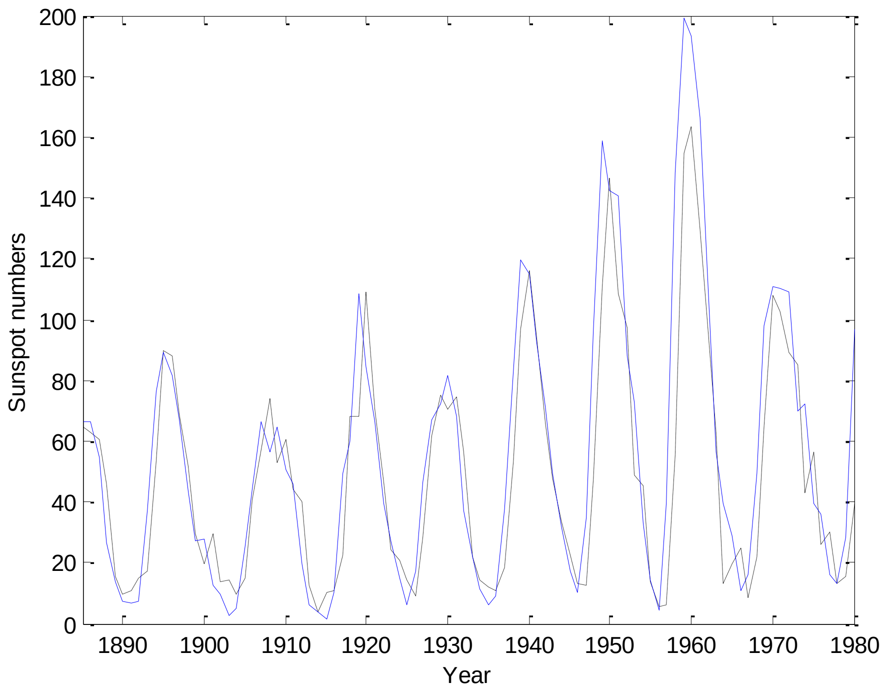


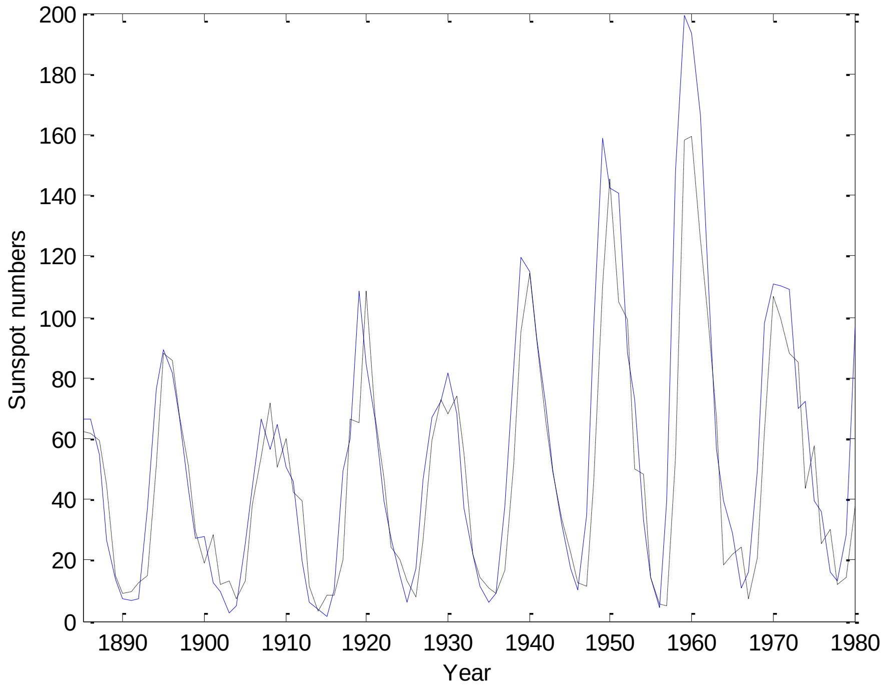
Fig. 5. Sunspot numbers from year 1700 to 1980.



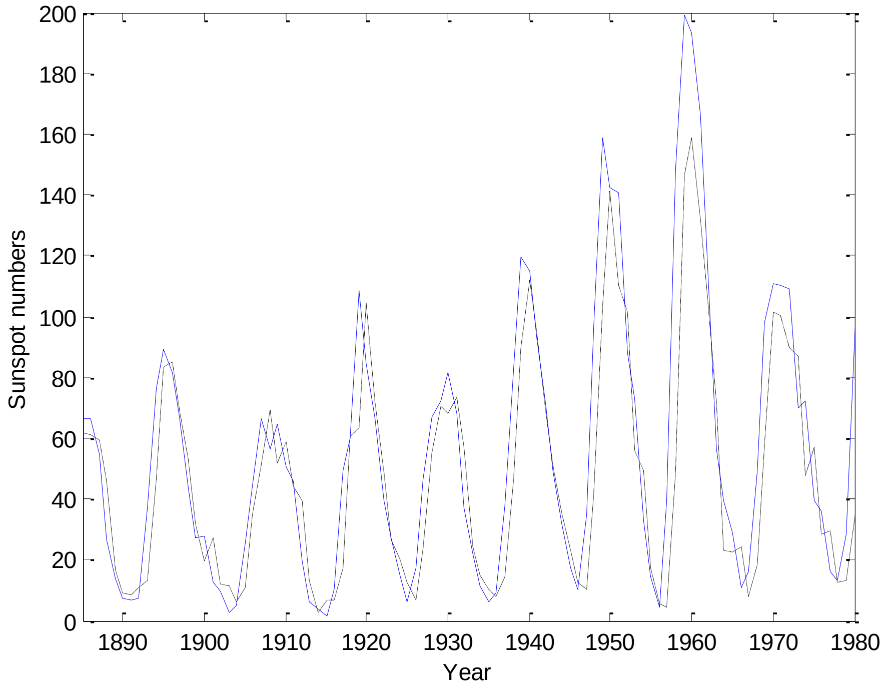
(a) Number of hidden nodes (n_h) = 4.



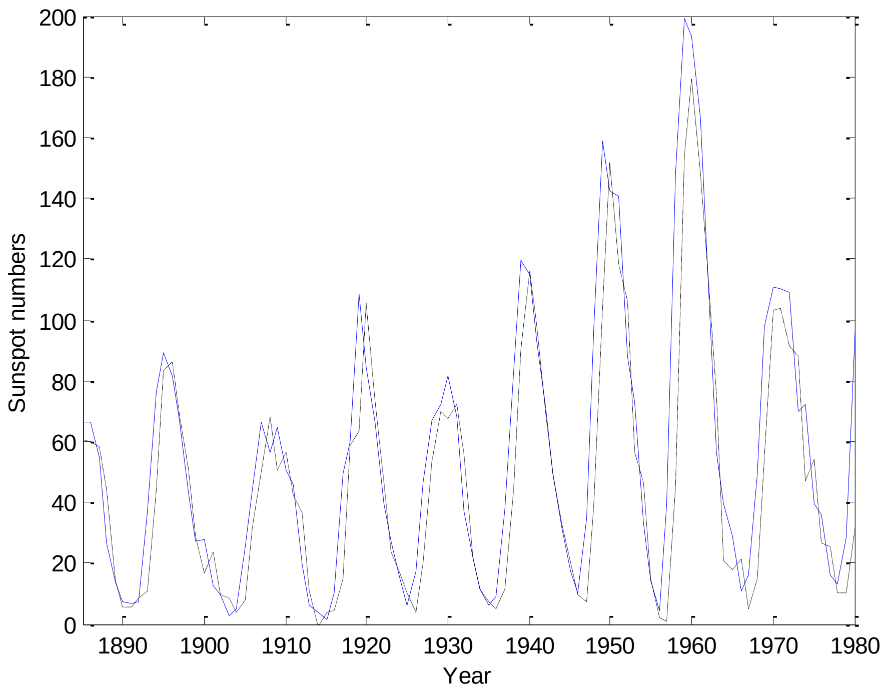
(b). Number of hidden nodes (n_h) = 5.



(c). Number of hidden nodes (n_h) = 6.



(d). Number of hidden nodes (n_h) = 7.



(e). Number of hidden nodes (n_h) = 8.

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

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

Table I. Simulation results of the proposed GA and the standard GA based on the benchmark test functions.

	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

(a)

	Standard GA with traditional neural network		Back-Propagation with Momentum and Adaptive Learning Rate	
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

(b)

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	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

(a)

	Standard GA with traditional neural network		Back-Propagation with Momentum and Adaptive Learning Rate	
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

(b)

Table III. Training error and forecasting error in mean absolute error (MAE) for the application example on forecasting the sunspot number.

	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

(a)

	Standard GA with traditional neural network		Back-Propagation with Momentum and Adaptive Learning Rate	
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

(b)

Table IV. Simulation results for the application example of associative memory after 500 iterations of learning.