Neurocomputing 86 (2012) 140–149

Contents lists available at SciVerse ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications

Shih-Hung Yang*, Yon-Ping Chen

Department of Electrical Engineering, National Chiao Tung University, EE609, 1001 University Road, Hsinchu 300, Taiwan

ARTICLE INFO

Article history:
Received 30 August 2011
Received in revised form
13 December 2011
Accepted 17 January 2012
Communicated by W. S. Hong
Available online 23 February 2012

Keywords:
Evolutionary algorithm
Neural network
Constructive
Pruning
Prediction

ABSTRACT

We propose a method for designing artificial neural networks (ANNs) for prediction problems based on an evolutionary constructive and pruning algorithm (ECPA). The proposed ECPA begins with a set of ANNs with the simplest possible structure, one hidden neuron connected to an input node, and employs crossover and mutation operators to increase the complexity of an ANN population. Additionally, cluster-based pruning (CBP) and age-based survival selection (ABSS) are proposed as two new operators for ANN pruning. The CBP operator retains significant neurons and prunes insignificant neurons on a probability basis and therefore prevents the exponential growth of an ANN. The ABSS operator can delete old ANNs with potentially complex structures and then introduce new ANNs with simple structures; thus, the ANNs are less likely to be trapped in a fully connected topology. The ECPA framework incorporates constructive and pruning approaches in an attempt to efficiently evolve compact ANNs. As a demonstration of the method, ECPA is applied to three prediction problems: the Mackey-Glass time series, the number of sunspots, and traffic flow. The numerical results show that ECPA makes the design of ANNs more feasible and practical for real-world applications.

Crown Copyright © 2012 Published by Elsevier B.V. All rights reserved.

1. Introduction

Many numerical algorithms to accurately predict the trends of time series in the future have been proposed, including the autocorrelation method [1], the covariance method [2], and grey theory [3]. Recently, to further improve the accuracy of time series prediction, investigators have focused on intelligent algorithms based on artificial neural networks (ANNs) due to their learning abilities and powerful prediction capability [4], [5].

ANNs were first developed to imitate biological neural systems and are organized into several interconnected simple processing units called neurons or nodes. ANNs are data-driven approaches that learn from examples, even when the input–output relationships are unknown [6]. Thus, ANNs can accurately solve problems without prior knowledge when sufficient observed data are supplied. This property is useful for evaluating numerous forecasting problems because acquiring data is easier than making good theoretical guesses about certain systems.

An important component of every ANN is architecture selection, which involves determining an appropriate architecture to accurately fit the underlying function described by the training data [7]. An architecture that is too large may precisely fit the training data but may provide poor generalization due to overfitting of the training data. Conversely, an architecture that is too small saves computational costs but may not possess sufficient processing ability to accurately approximate the underlying function. Therefore, architecture selection should consider both network complexity and goodness of fit.

For prediction purposes, it has been shown that a feedforward ANN with a single hidden layer is sufficient to achieve any desired accuracy [8]. In most applications, ANNs are fully connected, i.e., all inputs are fully connected to all hidden neurons. Numerous studies have shown that partially connected ANNs have better storage capability per connection than fully connected ANNs [9], [10]. Furthermore, partially connected ANNs can yield improved generalization capabilities with reduced cost in terms of hardware and processing time [11]. However, how to determine the optimal numbers of hidden neurons and connections remains an open question.

Among several algorithms for designing three-layered ANNs, the most frequently used algorithms are the constructive, pruning, and constructive-pruning algorithms. A constructive algorithm [12] starts with a minimal ANN architecture, a three-layered ANN with one hidden neuron. The algorithm adds hidden neurons to the minimal ANN, one-by-one, during the training phase. The advantage of the constructive algorithm is that the initial phase can simply set the number of hidden layers and neurons as one each. However,

* Corresponding author. Tel.: +886 3 571 2121 ext. 54406; fax: +886 3 5731585.
E-mail address: pshong.ece93g@nctu.edu.tw (S.-H. Yang).

0925-2312/$ - see front matter Crown Copyright © 2012 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2012.01.024

deciding when to add hidden neurons or connections and when to stop the addition process is difficult.

A pruning algorithm [13] starts with an oversized architecture and then deletes unnecessary hidden neurons or connections, either during training or upon convergence to a local minimum. Each iteration of the pruning algorithm determines which unit, i.e., which hidden neuron or connection, to prune via its relevance or significance. Several pruning criteria have been proposed, for example, sensitivity analysis [14] and magnitude-based pruning [15]. Sensitivity analysis is based on Taylor expansion and reflects the ways in which the derivatives of a performance function can be applied to quantify a system's response to unit perturbations. Magnitude-based pruning assumes that small weights are irrelevant. However, no criterion can be used to determine the initially oversized architecture for a given problem [12].

In the constructive algorithm, the architecture of the ANN may become oversized if the addition procedure is not appropriately stopped. A number of algorithms have attempted to combine constructive and pruning algorithms to solve the aforementioned problem [16], [17]. These constructive-pruning algorithms first estimate the number of hidden neurons and/or connections via a constructive method. A pruning method is then used to delete the inappropriate hidden neurons and/or connections to find a near-optimal architecture for a given problem. However, determining when to stop the pruning procedure is difficult [18].

Several researchers have developed methods for designing ANNs using evolutionary algorithms (EAs). EAs emerged as a biologically plausible approach for adapting various ANN parameters such as weight values and architectures [19]. Recently, several studies have been proposed to employ various EAs to prune NNs. Mantzaris et al. [20] pruned probabilistic neural network by genetic algorithm to minimize the number of diagnostic factors, and therefore minimized the number of input nodes and hidden layers. Curry and Morgan [21] proposed a modified feedforward neural network which is pruned and optimized by means of differential evolution for seasonal data. Huang and Du [22] use particle swarm optimization to prune the radial basis probabilistic neural networks. Masutti and Castro [23] combined characteristics from self-organizing networks and artificial immune systems to solve the traveling salesman problem and pruned neurons which are not related to a city. Furthermore, numerous works have been done to perform EAs and pruning methods separately or simultaneously. Kaylani et al. [24] incorporated prune operator into a genetic algorithm as a mutation operator to design ARTMAP architecture for classification problems. Goh et al. [25] developed a hybrid multiobjective evolutionary approach for adaptation of ANNs structures and a geometrical approach in identifying hidden neurons to prune for classification problems. Hervás-Martínez et al. [26] applied an evolutionary algorithm to design the structure and weights of a product-unit neural network, and finally used a backward stepwise procedure to prune variables sequentially until no further pruning can be made to improve the fit. However, most encoding schemes must predefine the chromosome length, which is problem-dependent. This user-defined length can affect the flexibility of problem representation and EA efficiency [27], [28].

Herein, we propose a new approach for designing ANNs, the evolutionary constructive and pruning algorithm (ECPA). This algorithm directs the evolution of the ANN topology using constructive and pruning methods in an evolutionary manner. In ECPA, a variable-length chromosome representation is adopted to describe ANNs with different architectures. Thus, it is not necessary to predefine the length of the chromosome, and this makes the use of memory more efficient. Furthermore, ECPA introduces the concept of constructive method into the crossover and mutation operations in a manner that allows the initial structure of the ANN to be simply set as a minimal network containing one hidden neuron with a single connection to one input. The crossover and mutation operations then enlarge the architecture by adding hidden neurons and connections. ECPA then prunes the resulting ANNs via a newly developed scheme consisting of cluster-based pruning (CBP) and age-based survival selection (ABSS).

The rest of this paper is organized as follows. Section 2 describes the proposed ECPA in detail. Section 3 demonstrates the proposed algorithm's ability to evolve partially connected ANNs for a variety of problems of interest. Finally, in Section 4, we present our conclusions.

2. ECPA

Based on the characteristics of ANNs and EA, we propose ECPA to develop ANNs based on an evolutionary constructive and pruning manner. As discussed in [29], theoretical work has shown that a single hidden layer is sufficient for forecasting purposes. Therefore, in this work, we designed a three-layer feedforward ANN with an input layer, a hidden layer, and an output layer. The major steps of ECPA are summarized in Fig. 1 and explained below.

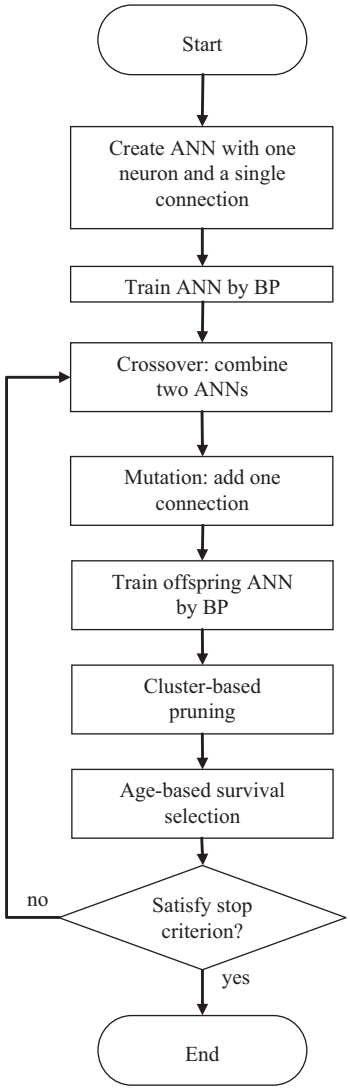

Fig. 1. Major steps performed in ECPA.

Initialization phase

(Step 1) Generate an initial population with N_p ANNs, where N_p is the population size. The initial ANN structure starts with a minimal network with one neuron and a single connection from one of the inputs, which is randomly selected.
(Step 2) Train all ANNs using backpropagation (BP) algorithm [30] for φ epochs, where φ is specified by the user, and determine their fitness.

Reproduction phase

(Step 3) Select two parents by tournament selection. Produce one offspring from the two parents using network crossover with crossover probability, p_c.
(Step 4) Apply network mutation to the offspring with mutation probability, p_m.
(Step 5) Train the offspring ANN using BP for φ epochs and determine its fitness.
(Step 6) Perform CBP on the offspring. Go to Step 3 until N_p offspring are generated.
(Step 7) Apply ABSS to the parents and their offspring to generate the parents of the next generation. Go to Step 3 until the maximum number of generations, G, is reached.
(Step 8) Select a single best ANN among the final population.

2.1. Encoding scheme

In order to encode an ANN into a chromosome, ANN is represented as a vector whose length depends on the size of ANN such that the memory can be used efficiently. Fig. 2 shows the chromosome representation of two ANNs and their corresponding graphical representations. The chromosome consists of the network connections and weights where w_1, w_{b1}, and w_{12} indicate the output weight of the first hidden neuron, the weight connected from bias, and weight connected between first hidden neuron and second input node, respectively. Note that the weight with nonzero value represents the connected weight while that with zero value represents the disconnected weight. The initial population is a set of simplest possible networks whose initial weights are randomly generated by a uniform distribution in the range $[-1.0, 1.0]$ via the suggestion in [31]. ECPA directs the evolution of the ANNs via four essential components: network crossover, network mutation, CBP, and ABSS. Details regarding each component of ECPA are provided in the following sections.

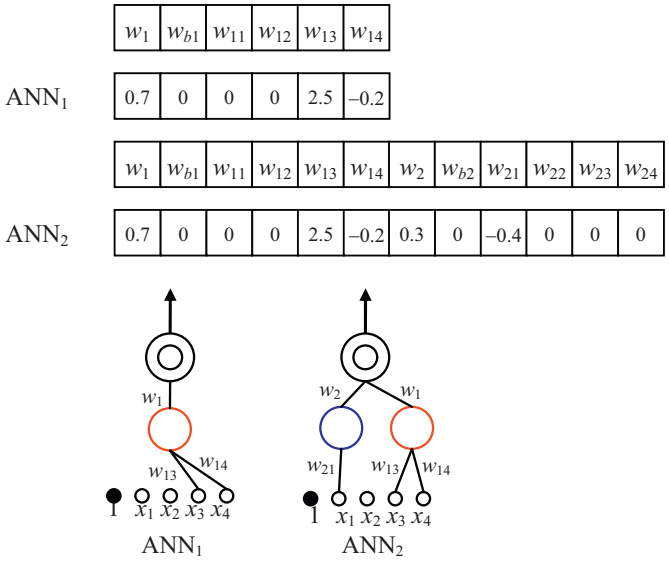

Fig. 2. Coding of NN and two examples.

2.2. Network crossover

The ECPA starts from a population of ANNs with the simplest possible structures so that the initialization phase can easily set the topology of ANNs as one hidden neuron and a single connection from one input. However, these ANNs may not be able to achieve sufficient and desired accuracy. In order to increase the processing capabilities of NNs, it is necessary to facilitate the exploration of the wider regions of a structural search space. For the sake of this objective, this stage executes constructive manner to add hidden neurons to each NN. To decide how many hidden neurons should be add to each NN, network crossover simply selects two ANNs and combines them together. Hence, this operator does not require many heuristics, user-defined parameters, and rich prior knowledge. The network crossover operation in ECPA produces an offspring ANN by combining the substructures of two parent ANNs. To clearly illustrate the network crossover, an example is shown in Fig. 3 for two parent ANNs, ANN_a and ANN_b, and their offspring ANN_c. The input–output relationship of ANN_a is as follows:

$$y^a = w^a_{o1} \cdot h(w^a_{h11} \cdot x_1 + w^a_{h12} \cdot x_2) \tag{1}$$

where h is the hidden node activation function, w^b_{o1} is the output weight, and w^a_{h12} is the weight connected from x_2 to the hidden node. The hidden node activation function can be a linear, logistic or hyperbolic tangent function. The superscript of each weights represents its network index, and the subscript indicates the relationship between neurons. For ANN_b, the input–output relationship is as follows:

$$y^b = w^b_{o1} \cdot h(w^b_{h11} \cdot x_1) + w^b_{o2} \cdot h(w^b_{h22} \cdot x_2 + w^b_{h24} \cdot x_4) \tag{2}$$

where w^b_{o1} is the output weight of the first hidden node and w^b_{h24} is the weight from x_4 to the second hidden node.

The network crossover directly combines the substructures of ANN_a and ANN_b, and the offspring ANN_c is subsequently obtained as:

$$\begin{aligned} y^c &= 1/2(y^a + y^b) \\ &= 0.5w^a_{o1} \cdot h(w^a_{h11} \cdot x_1 + w^a_{h12} \cdot x_2) \\ &\quad + 0.5w^b_{o1} \cdot h\left(w^b_{h11} \cdot x_1\right) + 0.5w^b_{o2} \cdot h(w^b_{h22} \cdot x_2 + w^b_{h24} \cdot x_4) \end{aligned} \tag{3}$$

This result is shown in Fig. 3. The output weights of the offspring ANN are half those of the parent ANNs, and the hidden weights of the offspring retain the same weights as those of the parent ANNs. As shown in Fig. 3, it is clear that the network crossover operator directs the evolution of the ANNs in a constructive manner. Furthermore, a crossover probability is

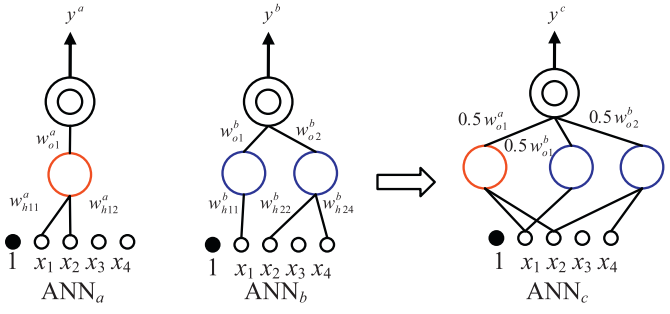

Fig. 3. An example of a network crossover.

chosen to determine whether or not to perform network crossover on two parent ANNs. If p_c is larger than a random number, network crossover is performed on the two parent ANNs; otherwise, the two parent ANNs are copied as two offspring.

2.3. Network mutation

When the network crossover is applied to parent ANNs, some offspring ANNs are likely to have more hidden neurons and thus possess much processing ability. However, it may be inefficient to increase ANN's performance by only adding hidden neuron with single connection generated by the initialization phase. It is possible to introduce more inputs into each hidden neuron to increase the prediction accuracy. EAs usually adopt mutation operator to achieve the perturbation and thus have a better exploitation capability. Hence, a small perturbation of structure is suitable for structural leaning. Since the initialization phase generates ANNs with single connection, a small perturbation can be achieved via adding more connections to ANNs and its capability is distinct from adding hidden neurons. For the sake of simple and small perturbation, network mutation introduces a new connection into ANN where the connection is built between randomly selected one input and one hidden neuron, and initializes its weight according to a uniform distribution in the range $[-0.01, 0.01]$. A graphical representation of this operation, where a new connection is added between x_2 and the first hidden node, is shown in Fig. 4. The input–output relationship of the mutated ANN_b, $\mathrm{ANN}_{b'}$, is written as follows:

$$y^b = w_{o1}^b \cdot h(w_{h11}^b \cdot x_1 + w_{h12}^b \cdot x_2) + w_{o2}^b \cdot h(w_{h22}^b \cdot x_2 + w_{h24}^b \cdot x_4) \quad (4)$$

where $w_{h12}^b = 0$ and thus, $\mathrm{ANN}_{b'}$ retains the performance of ANN_b. To further improve the performance of $\mathrm{ANN}_{b'}$, the training process executes Step 5.

2.4. CBP

ECPA employs network crossover and network mutation to design ANNs in a constructive manner. However, it is well known that the constructive algorithms difficultly decide when to stop the addition process and may design an excessively large and complex ANN with poor generalization performance. Thus, a pruning algorithm can be used to determine the relevance or significance of hidden neurons and delete insignificant ones. Nevertheless, it is not easy to determine the threshold value for distinguishing insignificant hidden neurons from significant ones. Therefore, ECPA uses a different pruning scheme, called CBP, which simply separates the hidden neurons into two classes, good and worse, according to the best hidden neuron and the worst one without any user specified parameter. Then the hidden neurons in worse class are pruned in a stochastic way to avoid deleting excessive hidden neurons. Unlike conventional pruning algorithms, CBP proceeds in three steps.

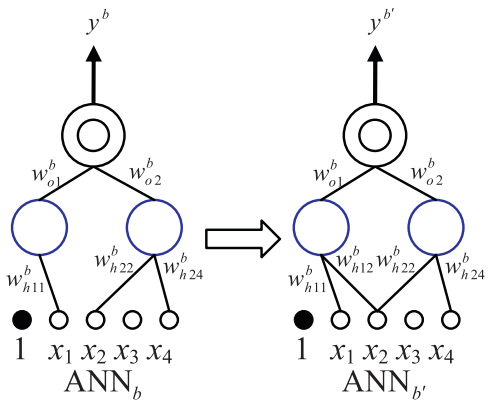

Fig. 4. An example of a network mutation.

In the first step, the significance of each hidden neuron is determined. For the ith hidden neuron, the significance is defined as

$$\sigma_i = \sqrt{S_i} \quad (5)$$

where S_i is obtained as follows [32], [33]:

$$S_i = \sqrt{\frac{\sum_{p=1}^{P} (S_i^p)^2}{P}} \quad (6)$$

Thus, S_i is the root-mean-square of S_i^p, which is the sensitivity of the network output o^p to the output h_i^p of the ith hidden neuron for the pth pattern, expressed as

$$S_i^p = \frac{\partial o^p}{\partial h_i^p} = w_i \quad (7)$$

Here, w_i is the weight of the connection from the ith hidden neuron to the output neuron; this weight is constant because it is irrelevant to the patterns. Hence, the significance in (5) can be rewritten as

$$\sigma_i = \sqrt{|w_i|} \quad (8)$$

Thus, a hidden neuron with low significance has little influence on the network output and can be removed [14]. However, to avoid excessive pruning of the hidden neurons, the significance in (5) is purposely chosen as the square root of S_i, following the concept of the rootogram [34].

In the second step, the hidden neurons are categorized into two classes: good and worse. The prototypes of the good and worse classes are the hidden neurons with maximum and minimum significance, respectively. The remaining hidden neurons are categorized according to the difference between the good and worse prototypes with respect to their significance. If the significance of one hidden neuron is close to the good prototype, the hidden neuron is then categorized in the good class; otherwise, the hidden neuron is categorized in the worse class. The neurons in the good class are retained, whereas those in the worse class are deleted in a stochastic manner. For each neuron in the worse class, a random number r with a uniform distribution between [0,1] is generated. If r is smaller than 0.5, the neuron is deleted; otherwise, the neuron is retained.

2.5. ABSS

After the network crossover, network mutation and CBP are completed, the individuals in the next generation are chosen through survival selection. If a general survival selection is adopted, the evolved ANNs tend to have fully connected topologies due to network mutations, which add more inputs to the hidden neurons. As a result, hardware implementation costs are increased, and the generalization capabilities of the evolved ANNs are reduced. To avoid this problem, we propose a different survival selection method, ABSS, to select younger ANNs with partial connections, rather than full connections, for the next generation.

ABSS is performed in two stages. The first stage involves traditional tournament selection to choose N_p candidates for the next stage. If the age of an ANN is defined as the number of generations it survives in the population, then the N_p candidates may have different ages. For example, the age of a newborn ANN is one, and its age increases by one if it survives to the next generation. The second stage continues to delete the elder ANNs from the N_p candidates according to the health index, defined as

follows:

$$H_j = \left(1 - \frac{1}{\mathrm{Age}_j}\right)^2 \tag{9}$$

where Age_j is the age of the jth ANN. Selection proceeds by generating a uniform random number r in the range [0,1]. If $H_j > r$, the jth ANN is deleted and replaced by a newborn ANN produced by Step 1; otherwise, the jth ANN is retained in the population. As a result, the population size N_p is unchanged after ABSS, and the average age of the ANNs is potentially lower, which prevents the evolved ANNs from adopting a fully connected topology.

In summary, the network crossover operator constructs an ANN by adding hidden neurons so that the ANN possesses more processing ability to accurately approximate the target function. The network mutation operator adds one connection from the input to the hidden neuron so that the hidden neuron can process more input information. CBP prunes the worse hidden neurons from an ANN to prevent overfitting of the training data. ABSS deletes elder ANNs that are potentially fully connected. Thus, network crossover and mutation operations direct the evolution of ANNs in a constructive manner that can improve their processing ability to accurately approximate the true function, whereas CBP and ABSS direct the evolution of ANNs in a destructive way that can improve their generalization capabilities while reducing their hardware requirements.

2.6. Computational complexity

The computational complexity of ECPA is dominated by BP algorithm. It has been known that the computational complexity of BP is of the order of $O(N_c)$ where N_c denotes the number of connections or weights [35]. Note that N_c is small due to the use of simplest possible ANNs in the beginning generations while N_c is large as applying the network crossover and network mutation in the later generations. Thus, the total computational complexity of ECPA can be easily derived as the order of $O(N_p \times G \times \varphi \times N_c)$. Clearly, the computational cost of large and complex ANN is more expensive than small and simple NN. Hence, CBP and ABSS are necessary for pruning ANNs and reducing computational cost.

3. Experimental results

In this section, we demonstrate the performance of the proposed algorithm using three time series prediction problems: Mackey-Glass, sunspots, and vehicle count. The first time series is generated from the Mackey-Glass differential equation, the second series is recorded from the sunspots, and the third series is obtained from the hourly vehicle count for the Monash Freeway outside Melbourne in Victoria, Australia, beginning in August, 1995. During the evolutionary process, the root-mean-square-error (RMSE) is adopted as the fitness, which is calculated as follows:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}(x(t) - \hat{x}(t))^2} \tag{10}$$

where $\hat{x}(t)$ is the predicted value at time t and N is the number of data points. To observe the evolutionary progress of an ANN, the RMSE, the number of hidden nodes N_h, and the number of connections N_c are recorded.

As described in Section 3.5, the N_p, G, and φ would affect the computational complexity of ECPA. The larger N_p the less effect of genetic drift, the larger G the more chances to find better ANNs, and the larger φ the more prediction accuracy. However, the larger N_p, G, and φ lead to the longer computation time. To select suitable values, N_p is set as 30 according to the suggestion in [36]. In order to select appropriate φ and G, the values of φ were chosen as 5, 10, 15, 20, 25, and 30, and the values of G were chosen as 300, 400, 500, and 600 in the preliminary runs. As a result, $G=500$ and $\varphi=15$ were adopted in the following experiments due to the sufficient prediction accuracy and acceptable computation time. Because the parameters were chosen after some preliminary runs, the values were not meant to be optimal. The setting of $p_c=0.8$ and $p_m=0.6$ is to provide more chances of increasing the number of hidden neurons than increasing the number of connections. It was expected that structures with more hidden neurons would be found first, and these structure would then be pruned.

3.1. Results on the Mackey-Glass time series

The Mackey-Glass time series prediction is recognized as a benchmark problem in the area of ANNs. This chaotic time series prediction was considered to be a suitable way to evaluate the performance of the proposed ECPA. The Mackey-Glass time series is generated from the following delay differential equation:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{11}$$

where $\tau=17$ and $x(0)=1.2$ in the simulation. The fourth-order Runge–Kutta method is used to generate 1000 data points ranging from $t=118$ to 1117. The task involves predicting the value of $x(t+6)$ from the input vector $[x(t-18)\ x(t-12)\ x(t-6)\ x(t)]$ for any t. Therefore, the input–output data pairs for prediction are

$$[x(t-18),x(t-12),x(t-6),x(t);x(t+6)]$$

where the first 500 data pairs are used as training data, and the last 500 data pairs are used as testing data.

For better understanding ECPA, the following statements are used to demonstrate the major steps of ECPA in detail. In Step 1, 30 ANNs are generated with one hidden neuron and a single connection from one of the four inputs. In Step 2, the 30 ANNs are trained by BP for 15 epochs and their fitness is determined according to (10). In Step 3, two parent ANNs are selected by tournament selection based on their fitness. If a random number is smaller than 0.8, one offspring ANN is produced from the two parent ANNs by network crossover; otherwise, the two parent ANNs are copied as two offspring ANNs. In Step 4, the offspring ANN is mutated by network mutation with $p_m=0.6$. In Step 5, the offspring ANN is trained by BP for 15 epochs and its fitness is determined according to (10). In Step 6, the significance of each neuron of the offspring ANN is calculated by (8). The neurons are categorized into good and worse classes by their significances and the neuron in worse class is deleted if a random number is smaller than 0.5. Then go to Step 3 until 30 offspring ANNs are generated. In Step 7, 30 ANNs are chosen by the traditional tournament selection from the parent and offspring ANNs. Then the health indices of the 30 ANNs are determined via (9). If the health index of an ANN is larger than a random number, the ANN is further deleted and replaced by a newborn ANN produced by Step 1. Then go to Step 3 until reach 500th generation. In Step 8, a single best ANN is selected as the solution of the problem.

The evolutionary progress of the ANNs for the Mackey-Glass time series prediction problem is illustrated in Fig. 5. The top panel of Fig. 5 shows the decrease in the RMSE resulting from the evolution of the ANNs. The middle and bottom panels of Fig. 5 present N_h and N_c, respectively and demonstrate the structural evolution of the ANNs. Fig. 6 graphically illustrates how the topologies of the ANNs evolve in selected generations. The input vector $[u(4)u(3)u(2)u(1)]$ represents $[x(t-18)x(t-12)x(t-6)x(t)]$, and the output y represents $x(t+6)$. The blue lines represent

positive-valued weights, and the red lines represent negative-valued weights. The widths of the lines indicate the relative strengths of the weights. The ANNs were observed to grow rapidly, but growth did not always occur due to the use of CBP and ABSS. Note that the resulting ANN structure does not have a fully connected topology; less than 85% of the synapses are connected.

Many approaches have been developed to design both the architecture and weights of ANNs to address the same prediction problem. Table 1 presents the average results obtained using the proposed algorithm and other algorithms. Note that the results of ECPA are averaged over 10 independent runs and thus N_h and N_c are decimal numbers. As shown here, although ECPA achieved a larger RMSE than that of Du and Zhang [37] with the training data, it obtained a lower RMSE than the other methods for the testing data. It is interesting that ECPA obtained a lower RMSE for the testing data than for the training data in this experiment, but this phenomenon has been observed previously [38]. In terms of the average number of hidden neurons, ECPA obtained a lower N_h than those of Du and Zhang [37] and Harpham and Dawson [39]. Although ECPA obtained a higher N_h than those of Rojas et al. [40], Chen et al. [41], and Cho and Wang [42], it achieved a lower RMSE. However, it took much longer computation time, T_c, than Chen et al. [41], which is the major cost in real problem. ECPA resulted in the evolution of an ANN with training data RMSE, testing data RMSE, N_h, and N_c values of 6.76×10^{-4}, 6.30×10^{-4}, 40.5, and 203.2, respectively. Clearly, the evolved ANN possessed a partially connected topology; our observations showed that ECPA can evolve ANNs with a lower RMSE and more compact structure than the other methods.

In addition to the one-step prediction of $x(t+6)$, the evolved ANN was applied to another general testing case: the multiple-step prediction of $x(t+84)$ [38]. To perform a multiple-step prediction, the proposed algorithm iteratively predicts $x(t+6)$, $x(t+12)$, etc. until it reaches $x(t+84)$ after 14 such iterations. As shown in Table 1, the prediction error the multiple-step prediction increases to 3.10×10^{-3} because multiple-step prediction is more complex than one-step prediction. However, the prediction results of ECPA were still superior to the other methods according to the prediction error of multiple-step prediction.

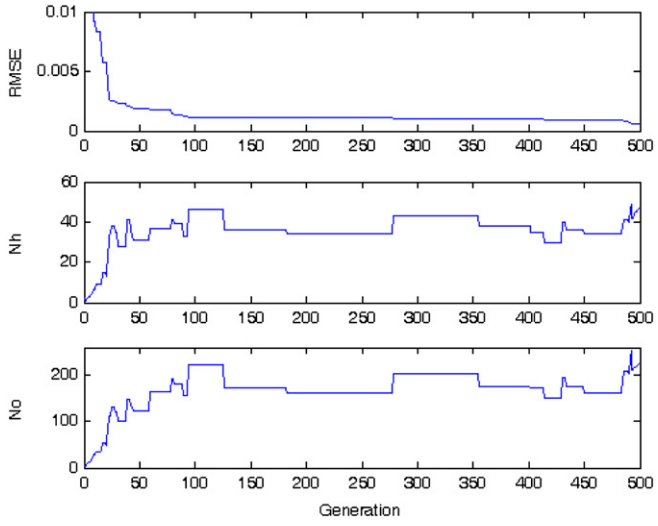

Fig. 5. Evolution progress for the Mackey-Glass time series.

3.2. Results on the number of sunspots

The number of sunspots varies nonlinearly in nonstationary and non-Gaussian cycles that are difficult to predict [43]. In this experiment, ECPA was used to predict the number of sunspots. The objective of this test involves using $[x(t-10)\ x(t-9)\ \ldots\ x(t)]$ to predict $x(t+1)$, where t represents the year and $x(t)$ represents the number of sunspots in year t. For a fair comparison with the other methods, the number of sunspots from 1700 to 1920 was used as the training data, and the number of sunspots from 1921 to 1955 was used as the testing data.

The learning curves of ECPA, including the training error, N_h, and N_c, in this example are shown in Fig. 7. To illustrate the structural evolution of the ANNs in detail, Fig. 8 shows the topologies of the ANNs in selected generations. The input vector $[u(11)\ u(10)\ \ldots\ u(1)]$ represents $[x(t-10)\ x(t-9)\ \ldots\ x(t)]$, and the output y represents $x(t+1)$. We observed that N_h was less than

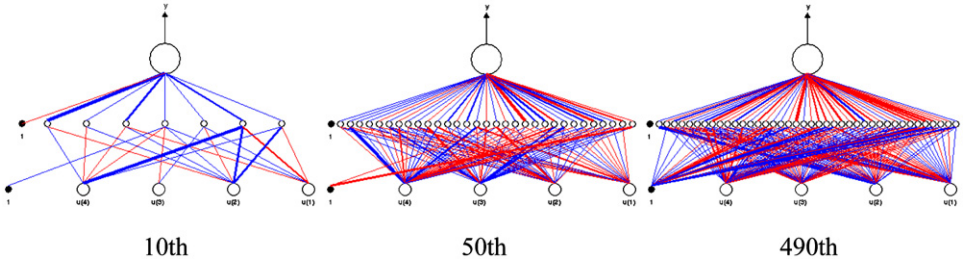

Fig. 6. Evolved ANNs for the Mackey-Glass time series.

Table 1
Prediction results for the Mackey-Glass time series.

Method	$\Delta t=6$		$\Delta t=84$		N_h	N_c	T_c
	Train	Test	First 500 points	Last 500 points			
Du and Zhang [37]	2.87×10^{-4}	7.67×10^{-4}	1.93×10^{-2}	2.07×10^{-2}	294	–	–
Harpham and Dawson [39]	–	1.50×10^{-3}	–	––	116	–	–
Rojas et al. [40]	2.87×10^{-3}	–	2.63×10^{-2}	–	12	–	–
Chen et al. [41]	3.30×10^{-3}	3.60×10^{-3}	–	–	10	110	114
Cho and Wang [42]	9.60×10^{-3}	1.14×10^{-2}	–	–	23	–	–
ECPA	6.76×10^{-4}	$\mathbf{6.30\times10^{-4}}$	$\mathbf{6.20\times10^{-3}}$	$\mathbf{3.10\times10^{-3}}$	40.5	203.2	1112.7

5 before the 12th generation, and only a few inputs were connected to the hidden neurons. N_h increased to 10 in the 40th generation and further increased to 12 in the 60th generation. During the 40th and 300th generations, the structures of the evolved ANNs grew rapidly, and more inputs were processed. Finally, N_h and N_c dropped to 5 and 32, respectively, at the end of the evolution process. Notably, all of the evolved ANNs lack connections from the bias of the hidden layer to the output layer. The final evolved ANN connected most of the inputs, except for $x(t-7)$, and each neuron connected an average of 5.6 inputs. Thus, the final evolved ANN clearly has a partially connected topology.

Table 2 presents the average performance of the evolved ANNs for the testing set over ten independent runs. ECPA performance was compared to those of an adaptive neural network (ADNN) [44], an artificial neural network (ANN) [4], a hybrid methodology that combines an autoregressive integrated moving average with an artificial neural network (Hybrid) [45], and an adaptive k-nearest neighbors (AKN) [46]. In terms of average performance, ECPA obtained an ANN with mean absolute percentage error (MAPE), normalized mean squared error (NMSE), N_h, and N_c values of 24.66, 0.0573, 5.0, and 46.9, respectively. These results indicate that the proposed ECPA can design an ANN with a compact structure and a smaller prediction error than other methods.

3.3. Results on the vehicle count

The vehicle count data set was obtained from the hourly vehicle count for the Monash Freeway outside Melbourne in Victoria, Australia, beginning in August 1995. The objective of this experiment involves using $[x(t-15)\ x(t-14)\ \ldots\ x(t)]$ to predict $x(t+1)$.

The top panel of Fig. 9 shows that NMSE of the evolved ANNs decreased rapidly in the first 50 generations and converged in the later generations. In addition to the middle and bottom panels of Fig. 9, the structural evolution of the ANNs is presented in more detail in Fig. 10, which graphically illustrates the topologies of the ANNs in selected generations. The 16 inputs $[u(16)\ u(15)\ \ldots\ u(1)]$ represent $[x(t-15)\ x(t-14)\ \ldots\ x(t)]$, and the output y represents $x(t+1)$. According to the middle panel of Fig. 9, N_h increased to 4 in the 4th generation and further increased to 9 in the 8th generation. After the 9th generation, N_h varied from 7 to 13. Finally, N_h converged to 7 in the 373rd generation. According to the bottom panel of Fig. 9, the ANNs have few connections in the early generations due to the single-connection topology present in the initial population, and N_c does not always increase. N_c drops from 37 to 22 in the 20th generation and from 40 to 37 in the 240th generation, implying that more connections do not necessarily result in superior fitness. In other words, appropriate topology, not the number of connections, is the key to improving

Table 2
Prediction results for the sunspot time series.

Method	MAPE	NMSE	N_h	N_c
ADNN [44]	28.45	0.068	6	–
ANN [4]	30.8	0.078	6	–
Hybrid [45]	31.2	0.0852	–	–
AKN [46]	50.3	0.1833	–	–
ECPA	**24.66**	**0.0573**	**5.0**	46.9

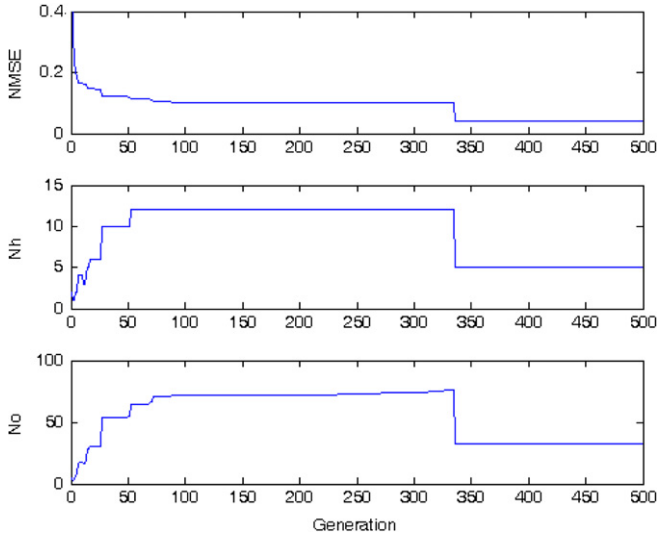

Fig. 7. Evolution progress for the sunspot time series.

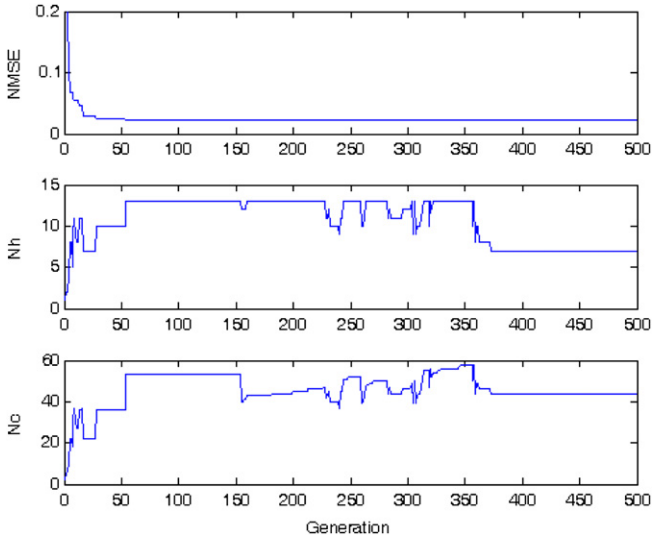

Fig. 9. Evolution progress for the vehicle count.

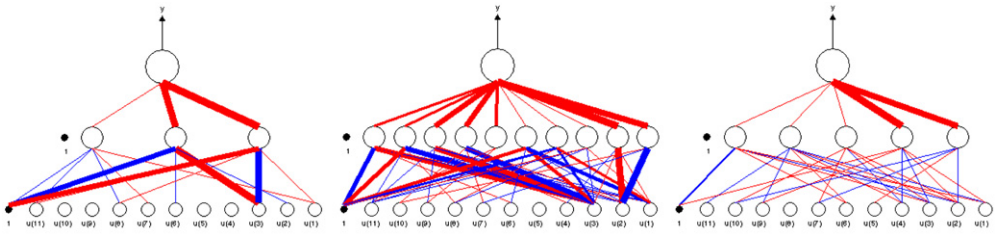

Fig. 8. Evolved ANNs for the sunspot time series.

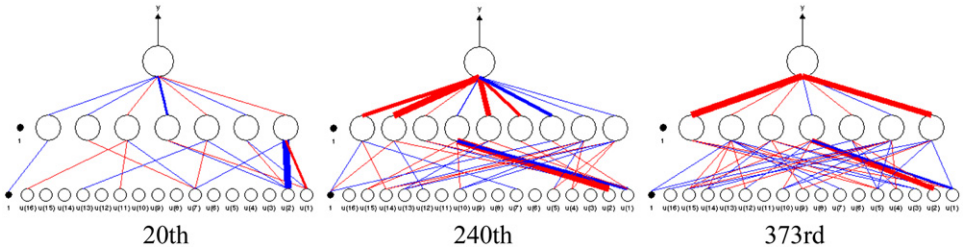

Fig. 10. Evolved ANNs for the vehicle count.

Table 3
Prediction results for the hourly vehicle count time series.

Method	MAPE	NMSE	N_i	N_h	N_c
ADNN [44]	14.31	0.0193	180	12	–
ANN [4]	17.97	0.0267	180	12	–
Hybrid [45]	26.98	0.0818	180	–	–
AKN [46]	17.39	0.0206	180	–	–
ECPA	**11.35**	**0.0182**	**16**	**7.7**	77.6

fitness. However, we observed that each neuron attempted to connect to more inputs as the number of generations was increased. The evolution of ANNs almost converges, and no further improvement in the ANNs was observed after the 373rd generation. As a result, the highest-quality ANN with well-trained weights has 7 hidden neurons and connects most of the inputs, except for $x(t-5)$ and $x(t-10)$. Thus, the ANN can automatically select the necessary inputs via ECPA. Clearly, the evolved ANN is a partially connected network.

Table 3 summarizes the average performance of the evolved ANNs for the testing set over ten independent runs and compares these results to other methods, including ADNN [44], ANN [4], Hybrid [45], and AKN [46]. The average NMSE using ECPA was 0.0182, which is less than that obtained using the other methods. Furthermore, the evolved ANNs obtained using ECPA have an average of 7.7 hidden neurons, which is less than the other methods.

3.4. Effect of CBP and ABSS

The previous section discusses the performance of ECPA for different prediction problems. However, the effect of CBP and ABSS on evolution of ANNs is unclear. To evaluate how CBP and ABSS affect ANN evolution, two variants of ECPA which do not use CBP and ABSS, respectively, were used in repetitions of the above experiments. The variant of ECPA without CBP is referred to as ECPAWC and that without ABSS is referred to as ECPAWA. The setup of these experiments was identical to those in the previous experiments.

In order to gain the deeper understanding of the performance difference between ECPA, ECPAWC, and ECPAWA in these three experiments, the three algorithms are compared in terms of NMSE, N_h, N_c, connection ratio, R_c, and computation time, T_c, whose unit is second. The R_c is determined as follows:

$$R_c = N_c/N_f \cdot 100\% \tag{12}$$

where N_f is the number of connections in an ANN with a fully connected topology. An ANN with N_h hidden nodes with a fully connected topology leads to the following relation:

$$\begin{aligned} N_f &= (N_i+1)\cdot N_h + N_h + 1 \\ &= (N_i+2)\cdot N_h + 1 \end{aligned} \tag{13}$$

where N_i is the number of input nodes. When $R_c < 100\%$, ANN has a partially connected topology, and when $R_c = 100\%$, ANN has a fully connected topology. The computational environment is Windows XP with Intel Core i7 870 2.93G CPU and 4GB RAM. These algorithms are implemented in MATLAB.

Table 4
Performance of ECPA and ECPAWA in Mackey-Glass, sunspot, and vehicle count time series.

Method	Experiment	NMSE	N_h	N_c	R_c (%)	T_c
ECPA	Mackey-Glass	$*6.3027 \times 10^{-4}$	40.5	203.2	83.28	1112.7
	Sunspot	0.0573	5.0	46.9	71.06	375.1
	Vehicle count	0.0182	7.7	77.6	55.59	693.3
ECPAWC	Mackey-Glass	$*1.3535 \times 10^{-3}$	860.6	2357.7	45.65	60531.7
	Sunspot	0.5741	419.3	1145.6	21.01	5841.3
	Vehicle count	0.0358	224.3	605.1	14.98	5939.6
ECPAWA	Mackey-Glass	$*6.3147 \times 10^{-4}$	575.3	2173.9	62.96	12958.9
	Sunspot	0.7547	173.7	603.1	26.70	2237.1
	Vehicle count	0.0176	70.8	653.6	51.25	3358.3

All results are averaged over 10 independent runs, where * refers to the RMSE.

The results in Table 4 present that ECPA and ECPAWA produce different ANNs in some aspects. For the three examples, the average N_h and N_c values over 10 independent runs returned by ECPAWA are much larger than those of ECPA which applies both CBP and ABSS. As comparing their prediction performance, ECPAWA yielded slightly smaller NMSE values than ECPA in the Mackey-Glass and vehicle count time series. This improvement may be yielded due to the great processing capability of a large number of hidden neurons. However, the ANNs developed via ECPAWA for the sunspot time series have larger NMSE value than those via ECPA. This may be due to the overfitting property caused by too many hidden neurons. The results indicate that ECPA facilitates ANNs with more generalization ability than ECPAWA.

In addition to NMSE, N_h, and N_c, Table 4 presents that R_c obtained by ECPAWC is lower than ECPAWA due to the use of ABSS. More specifically, elder ANNs which are much likely to have more connections from inputs caused by network mutation would be deleted by ABSS in ECPAWC. Although ECPAWC can produce sparsely connected topology of ANN by the aid of ABSS, it would result in ANNs with huge N_h. Thus, the ANNs in ECPAWC face overfitting problem and have bad generalization ability, i.e., larger NMSE for testing set. When comparing ECPA, ECPAWC, and ECPAWA, ECPAWA can produce ANNs with less hidden neurons than ECPAWC due to the use of CBP. ECPA further yields ANNs with more compact structures and better generalization ability than ECPAWA due to the use of ABSS.

Furthermore, the computation time of ECPA, ECPAWC, and ECPAWA is compared and shown in Table 4. Since N_c obtained by ECPAWC and ECPAWA is larger than ECPA, the computational costs of ECPAWC and ECPAWA are relatively higher than ECPA according to the computational complexity $O(N_p \times G \times \varphi \times N_c)$ described in Section 3.5. It is reasonable that the computation time through the overall learning process required by ECPA was less than that required by ECPAWC and ECPAWA. According to the

observation, both CBP and ABSS are beneficial for producing ANNs with a compact structure and reducing computation time.

3.5. Discussion

In this section, we summarize the observations in the three experiments described above, and discuss the experimental results. Figs. 5, 7 and 9 show that the ANN structures developed using ECPA are simple in the first generations due to the use of ANNs with one hidden neuron and a single connection to one of the inputs in the initial population. As their evolution progresses, the ANN structures grew rapidly due to the addition of neurons via crossover and the addition of connections via mutation. However, the ANNs do not grow continuously due to the use of CBP and ABSS. CBP primarily preserves the significant neurons and prunes the insignificant neurons using a probability criterion. Pruning prevents the exponential growth of the ANNs and avoids long-term training for complex ANNs. In addition, ABSS first deletes the old individuals likely to have complex structures and then provides an opportunity to introduce new individuals with simple structures generated via Step 1). Section 4.4 demonstrates that ABSS is useful for developing a compact ANN architecture and avoiding the design of complex ANNs. The highest-quality ANN with well-trained weights is then attained using construction via crossover and mutation operations and destruction via CBP and ABSS. The resulting ANNs do not have fully connected topologies; less than 80% of the synapses are connected in the Mackey-Glass time series, 50% are connected in the sunspot time series, and 40% are connected in the vehicle count time series. Furthermore, the evolved ANNs do not connect to all the inputs, e.g., the evolved ANN does not connect to $x(t-7)$ in the sunspot time series, and the evolved ANN does not connect to $x(t-5)$ and $x(t-10)$ in the vehicle count time series. Thus, ECPA has the ability to select the inputs required to accurately perform predictions. These results imply that more connections do not necessarily result in superior fitness. In other words, an appropriately connected topology is the key to improving fitness, not the number of connections.

4. Conclusions

A novel structural learning algorithm, called ECPA, is proposed for the design of ANNs based on an evolutionary constructive and pruning algorithm. ECPA evolves the ANNs starting with a minimal structure: one hidden neuron connected to an input node. The crossover and mutation operations make the ANN structures more complex, whereas CBP and ABSS make the ANN structures more compact. The results of the numerical simulations show that the use of CBP and ABSS operations indeed generates compact ANNs. Moreover, ECPA adopts variable-length chromosomes to represent the ANNs so that memory is used efficiently. In the time series prediction problems, ECPA not only evolved partially connected ANNs with sufficient prediction accuracy but also demonstrated the ability to select the proper inputs. The numerical results demonstrate that an appropriately connected topology, rather than the number of connections, is the key to improving ANN performance.

There are three future research directions suggested by this paper for the improvement of ECPA. First, the computational cost of ECPA is still expensive according to Table 4 due to the use of BP. Therefore, a scheme to make the weight training algorithm more efficient and reduce the computational complexity of ECPA could be investigated. Second, the two user-specified parameters, i.e., p_c, and p_m, could be devised as self-adaptive to increase the hidden neurons and connections more efficiently. Furthermore, it would be of great interest to make N_p self-adaptive so that N_p could be sometimes small and thus the computational complexity would be reduced. Third, ECPA has been applied to prediction problems. ECPA adopts BP as the weight training algorithm; however, BP may be not applicable in certain control problems when gradient information of the plants is not available. It would be interesting to study how to cooperate reinforcement learning into ECPA to design both parameter and structure for nonlinear control problems.

Acknowledgments

This work was supported in part by the National Science Council, Taiwan, R.O.C., under Contract No. NSC 99-2221-E-009-107 and in part by a grant provided by the Industrial Technology Research Institute under Contract No. A353C40000B1-4.

References

[1] J.D. Markel, A.H. Gray, Linear Prediction of Speech, Springer Verlag, New York, 1976.
[2] J. Makhoul, Linear prediction: a tutorial review, Proc. IEEE 63 (1975) 561–580.
[3] J.L. Deng, Introduction to grey system theory, J. Grey Syst. 1 (1989) 1–24.
[4] M. Adya, F. Collopy, How effective are neural networks at forecasting and prediction? A review and evaluation, J. Forecast. 17 (1998) 481–495.
[5] S.H. Yang, Y.P. Chen, Intelligent forecasting system using grey model combined with neural network, Int. J. Fuzzy Syst. 13 (2011) 8–15.
[6] B. Kosko, Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence, Prentice-Hall, Upper Saddle River, NJ, 1992 ch2.
[7] S. Kang, C. Isik, Partially connected feedforward neural networks structured by input types, IEEE Trans. Neural Networks 16 (2005) 175–184.
[8] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.
[9] S.T. Chen, D.C. Yu, A.R. Moghaddamjo, Weather sensitive short-term load forecasting using nonfully connected artificial neural network, IEEE Trans. Power Syst. 7 (1991) 1098–1105.
[10] A. Canning, E. Gardner, Partially connected models of neural networks, J. Phys. A 21 (1988) 3275–3284.
[11] D. Elizondo, E. Fiesler, A survey of partially connected neural networks, Int. J. Neural Syst. 8 (1997) 535–558.
[12] T.Y. Kwok, D.Y. Yeung, Constructive algorithms for structure learning in feedforward neural networks for regression problems, IEEE Trans. Neural Networks 8 (1997) 630–645.
[13] R. Reed, Pruning algorithms—A survey, IEEE Trans. Neural Networks 4 (1993) 740–747.
[14] A.P. Engelbrecht, A new pruning heuristic based on variance analysis of sensitivity information, IEEE Trans. Neural Networks 12 (2001) 1386–1399.
[15] J. Sietsma, R.J.F. Dow, Creating artificial neural networks that generalize, Neural Networks 4 (1991) 67–79.
[16] Y. Hirose, K. Yamashita, S. Hijiya, Back-propagation algorithm which varies the number of hidden units, Neural Networks 4 (1991) 61–66.
[17] I. Rivals, L. Personnaz, Neural-network construction and selection in nonlinear modeling, IEEE Trans. Neural Networks 14 (2003) 804–819.
[18] Md.M. Islam, Md.A. Sattar, Md.F. Amin, X. Yao, K. Murase, A new adaptive merging and growing algorithm for designing artificial neural networks, IEEE Trans. Syst. Man Cybern. Part B 39 (2009) 705–722.
[19] P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Trans. Neural Networks 5 (1994) 54–65.
[20] D. Mantzaris, G. Anastassopoulos, A. Adamopoulos, Genetic algorithm pruning of probabilistic neural networks in medical disease estimation, Neural Networks 24 (2011) 831–835.
[21] B. Curry, P.H. Morgan, Seasonality and neural networks: a new approach, Int. J. Metaheurist. 1 (2010) 181–197.
[22] D.-S. Huang, J.-X. Du, A constructive hybrid structure optimization methodology for radial basis probabilistic neural networks, IEEE Trans. Neural Networks 19 (2008) 2099–2115.
[23] T.A.S. Masutti, L.N. de Castro, Neuro-immune approach to solve routing problems, Neurocomputing 72 (2009) 2189–2197.
[24] A. Kaylani, M. Georgiopoulos, M. Mollaghasemi, G.C. Anagnostopoulos, AG-ART: an adaptive approach to evolving ART architectures, Neurocomputing 72 (2009) 2079–2092.
[25] C.-K. Goh, E.-J. Teoh, K.C. Tan, Hybrid multiobjective evolutionary design for artificial neural networks, IEEE Trans. Neural Networks 19 (2008) 1531–1548.

[26] C. Hervás-Martínez, F.J. Martínez-Estudillo, M. Carbonero-Ruz, Multilogistic regression by means of evolutionary product-unit neural networks, Neural Networks 21 (2008) 951–961.
[27] P.P. Palmes, T. Hayasaka, S. Usui, Mutation-based genetic neural network, IEEE Trans. Neural Networks 16 (2005) 587–600.
[28] T.B. Ludermir, A. Yamazaki, C. Zanchettin, An optimization methodology for neural network weights and architectures, IEEE Trans. Neural Networks 17 (2006) 1452–1459.
[29] G. Zhang, B.E. Patuwo, M.Y. Hu, Forecasting with artificial neural networks: the state of the art, Int. J. Forecast. 14 (1998) 35–62.
[30] D.E. Rumelhart, G.E. Hinton, R.J. Wiliams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[31] C. Zanchettin, T.B. Ludermir, L.M. Almeida, Hybrid training method for mlp: optimization of architecture and training, IEEE Trans. Syst. Man Cybern. Part B 41 (2011) 1097–1109.
[32] J.M. Zurada, A. Malinowski, I. Cloete, Sensitivity analysis for minimization of input data dimension for feedforward neural network, ISCAS (1994) 447–450.
[33] J.M. Zurada, A. Malinowski, S. Usui, Perturbation method for deleting redundant inputs of perceptron networks, Neurocomputing 14 (1997) 177–193.
[34] J.W. Tukey, Exploratory Data Analysis, Addison Wesley, Reading, MA, 1977.
[35] Z.H. Khan, T.S. Alin, Md.A. Hussain, Price prediction of share market using artificial neural network (ANN), Int. J. Comput. Appl. 22 (2011) 42–47.
[36] A.D. Cioppa, C.D. Stefano, A. Marcelli, On the role of population size and niche radius in fitness sharing, IEEE Trans. Evol. Comput. 8 (2004) 580–592.
[37] H. Du, N. Zhang, Time series prediction using evolving radial basis function networks with new encoding scheme, Neurocomputing 71 (2008) 1388–1400.
[38] J.S.R. Jang, ANFIS: adaptive-network-based fuzzy inference system, IEEE Trans. Syst. Man Cybern. 23 (1993) 665–685.
[39] C. Harpham, C.W. Dawson, The effect of different basis functions on a radial basis function network for time series prediction: a comparative study, Neurocomputing 69 (2006) 2161–2170.
[40] I. Rojas, H. Pomares, J.L. Bernier, J. Ortega, B. Pino, F.J. Pelayo, A. Prieto, Time series analysis using normalized PG-RBF network with regression weights, Neurocomputing 42 (2002) 267–285.
[41] Y. Chen, B. Yang, J. Dong, Time-series prediction using a local linear wavelet neural network, Neurocomputing 69 (2006) 449–465.
[42] K.B. Cho, B.H. Wang, Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction, Fuzzy Sets Syst. 83 (1996) 325–339.
[43] F.H.F. Leung, H.K. Lam, S.H. Ling, P.K.S. Tam, Tuning of the structure and parameters of a neural network using an improved genetic algorithm, IEEE Trans. Neural Networks 14 (2003) 79–88.
[44] W.K. Wong, M. Xia, W.C. Chu, Adaptive neural network model for time-series forecasting, Eur. J. Oper. Res. 207 (2010) 807–816.
[45] G.P. Zhang, Time series forecasting using a hybrid ARIMA and neural network model, Neurocomputing 50 (2003) 159–175.
[46] M. Kulesh, M. Holschneider, K. Kurennaya, Adaptive metrics in the nearest neighbours method, Phys. D 237 (2008) 283–291.

Shih-Hung Yang received his B.S. degree in Mechanical Engineering and M.S. degree in Electrical and Control Engineering from National Chiao Tung University, Taiwan, in 2002 and 2004, respectively. He is currently working toward the Ph.D. degree in Electrical and Control Engineering at National Chiao Tung University, Taiwan. His researches include machine vision, neural networks, and evolution computation. He was a recipient of the Outstanding Teaching Assistant Award from the ECE Department at National Chiao Tung University in 2008 and 2009, and Student Scholarships from IEEE Industrial Electronics Society and IEEE Computational Intelligence Society in 2010 and 2011, respectively.

Yon-Ping Chen received his B.S. degree in Electrical Engineering from National Taiwan University, Taiwan, in 1981, and M.S. and Ph.D. degrees in Electrical Engineering from University of Texas at Arlington, USA, in 1986 and 1989, respectively. He is a Distinguished Professor in the Department of Electrical Engineering, National Chiao Tung University, Taiwan. His researches include control, image signal processing, and intelligent system design.