ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Information Sciences xxx (2004) xxx–xxx

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

Time-series forecasting using flexible neural tree model

Yuehui Chen [a,*], Bo Yang [a], Jiwen Dong [a], Ajith Abraham [a,b]

[a] *School of Information Science and Engineering, Jinan University, 106 Jiwei Road, Jinan 250022, PR China*
[b] *School of Computer Science and Engineering, Chung-Ang University, Seoul, Republic of Korea*

Received 30 June 2003; received in revised form 19 October 2004; accepted 19 October 2004

Abstract

Time-series forecasting is an important research and application area. Much effort has been devoted over the past several decades to develop and improve the time-series forecasting models. This paper introduces a new time-series forecasting model based on the flexible neural tree (FNT). The FNT model is generated initially as a flexible multi-layer feed-forward neural network and evolved using an evolutionary procedure. Very often it is a difficult task to select the proper input variables or time-lags for constructing a time-series model. Our research demonstrates that the FNT model is capable of handing the task automatically. The performance and effectiveness of the proposed method are evaluated using time series prediction problems and compared with those of related methods.
© 2004 Published by Elsevier Inc.

Keywords: Flexible neural tree model; Probabilistic incremental program evolution; Simulated annealing; Time-series forecasting

[*] Corresponding author.
E-mail addresses: yhchen@ujn.edu.cn (Y. Chen), yangbo@ujn.edu.cn (B. Yang), csmaster@ujn.edu.cn (J. Dong), ajith.abraham@ieee.org (A. Abraham).

0020-0255/$ - see front matter © 2004 Published by Elsevier Inc.
doi:10.1016/j.ins.2004.10.005

1. Introduction

Artificial neural networks (ANNs) have been successfully applied to a number of scientific and engineering fields in recent years, i.e., function approximation, system identification and control, image processing, time series prediction and so on [1,36–39]. A neural network's performance is highly dependent on its structure. The interaction allowed between the various nodes of the network is specified using the structure. An ANN structure is not unique for a given problem, and there may exist different ways to define a structure corresponding to the problem. Depending on the problem, it may be appropriate to have more than one hidden layer, feedforward or feedback connections, or in some cases, direct connections between input and output layer.

There have been a number of attempts to design neural network architectures automatically. The early methods include constructive and pruning algorithms [2–4]. The main disadvantage of these methods is that the topological subsets are often searched using structural hill climbing methods instead of the complete class of ANNs architecture available in the search space [5]. Recent tendencies to optimize ANN architecture and weights include EPNet [6–8]and the NeuroEvolution of Augmenting Topologies (NEAT) [9]. Utilizing a tree to represent a NN-like model is motivated by the work of Byoung-Tak Zhang, where a method of evolutionary induction of the sparse neural trees was proposed [10]. Based on the representation of neural tree, architecture and weights of higher order sigma–pi neural networks were evolved by using genetic programming and breeder genetic algorithm, respectively.

Time-series forecasting is an important research and application area. Much effort has been devoted over the past several decades to develop and improve the time-series forecasting models. Well established time series models include: (1) linear models, e.g., moving average, exponential smoothing and the autoregressive integrated moving average (ARIMA); (2) nonlinear models, e.g., neural network models and fuzzy system models. Recently a tendency for combining of linear and nonlinear models for forecasting time series has been an active research area [11].

In this paper, a general and enhanced flexible neural tree (FNT) model is proposed for time-series forecasting problem. Based on the pre-defined instruction/operator sets, a flexible neural tree model can be created and evolved. This framework allows input variables selection, over-layer connections and different activation functions for different nodes. The hierarchical structure is evolved using probabilistic incremental program evolution algorithm (PIPE) [12,13] with specific instructions. The fine tuning of the parameters encoded in the structure is accomplished using simulated annealing (SA). The proposed method interleaves both optimizations. Starting with random structures and corresponding parameters, it first tries to improve the structure and then as soon as an improved structure is found, it fine tunes its parameters. It then goes back to improving the structure again and, fine tunes the structure and rules' parameters. This loop continues until a satisfactory solution is found or a time limit is reached.

The paper is organized as follows: Section 2 gives the representation and calculation of the flexible neural tree model. A hybrid learning algorithm for evolving the neural tree models is given in Section 3. Section 4 presents some simulation results for two time-series forecasting problems. Some concluding remarks are presented in Section 5.

2. Encoding and evaluation

In this research, a tree-structural based encoding method with specific instruction set is selected for representing a FNT model. The reason for choosing the representation is that the tree can be created and evolved using the existing or modified tree-structure-based approaches, i.e., genetic programming (GP) [41], probabilistic incremental program evolution (PIPE) [12], ant programming (AP) etc.

2.1. Flexible neuron instructor

The used function set F and terminal instruction set T for generating a FNT model are described as follows:

$$S = F \cup T = \{+_2, +_3, \ldots, +_N\} \cup \{x_1, \ldots, x_n\} \tag{1}$$

where $+_i$ ($i = 2, 3, \ldots, N$) denote non-leaf nodes' instructions and taking i arguments. $x_1, x_2, \ldots, x_n$ are leaf nodes' instructions and taking no other arguments. The output of a non-leaf node is calculated as a flexible neuron model (see Fig. 1). From this point of view, the instruction $+_i$ is also called a flexible neuron operator with i inputs.

In the creation process of neural tree, if a non-terminal instruction, i.e., $+_i$ ($i = 2, 3, 4, \ldots, N$) is selected, i real values are randomly generated and used for representing the connection strength between the node $+_i$ and its children. In addition, two adjustable parameters a_i and b_i are randomly created as

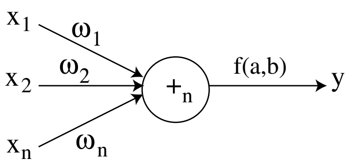


Fig. 1. A flexible neuron operator.

flexible activation function parameters. In this study the flexible activation function used is

$$f(a_i, b_i, x) = e^{-\left(\frac{x-a_i}{b_i}\right)^2} \tag{2}$$

The output of a flexible neuron $+_n$ can be calculated as follows. The total excitation of $+_n$ is

$$\mathrm{net}_n = \sum_{j=1}^{n} w_j * x_j \tag{3}$$

where x_j ($j = 1, 2, \ldots, n$) are the inputs to node $+_n$. The output of the node $+_n$ is then calculated by

$$\mathrm{out}_n = f(a_n, b_n, \mathrm{net}_n) = e^{-\left(\frac{\mathrm{net}_n - a_n}{b_n}\right)^2} \tag{4}$$

A typical flexible neural tree model is shown as Fig. 2. The overall output of flexible neural tree can be computed from left to right by depth-first method, recursively.

2.2. Fitness function

A fitness function maps FNT to scalar, real-valued fitness values that reflect the FNT's performances on a given task. Firstly the fitness functions should be seen as error measures, i.e., MSE or RMSE. A secondary non-user-defined objective for which algorithm always optimizes FNTs is the size of FNT usually measured by number of nodes. Among FNTs having equal fitness values

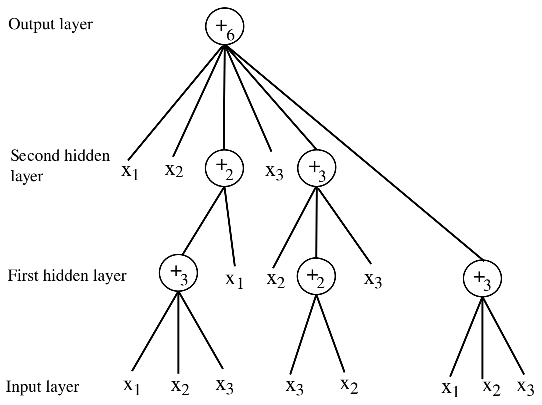


Fig. 2. A typical representation of neural tree with function instruction set $F = \{+_2, +_3, +_4, +_5, +_6\}$, and terminal instruction set $T = \{x_1, x_2, x_3\}$.

smaller FNTs are always preferred. In this work, the fitness function used for the PIPE and SA is given by mean square error (MSE):

$$\mathrm{Fit}(i) = \frac{1}{P}\sum_{j=1}^{P}(y_1^j - y_2^j)^2 \tag{5}$$

or root mean squared error (RMSE):

$$\mathrm{Fit}(i) = \sqrt{\frac{1}{P}\sum_{j=1}^{P}(y_1^j - y_2^j)^2} \tag{6}$$

where P is the total number of samples, y_1^j and y_2^j are the actual time-series and the FNT model output of jth sample. Fit(i) denotes the fitness value of ith individual.

3. A hybrid learning algorithm

In this study, finding an optimal or near-optimal neural tree structure is accomplished by using PIPE algorithm and the parameters embedded in a FNT is optimized by a SA [40].

3.1. Evolving an optimal or near-optimal neural tree structure

PIPE combines probability vector coding of program instructions, population-based incremental learning [14], and tree-coded programs. PIPE iteratively generates successive populations of functional programs according to an adaptive probability distribution, represented as a probabilistic prototype tree (PPT), over all possible programs. Each iteration uses the best program to refine the distribution. Thus, the structures of promising individuals are learned and encoded in the PPT.

The PPT stores the knowledge gained from experiences with programs (trees) and guides the evolutionary search. It holds the probability distribution over all possible programs that can be constructed from a predefined instruction set. The PPT is generally a complete n-ary tree with infinitely many nodes, where n is the maximal number of function arguments.

Each node N_j in PPT, with $j \geqslant 0$ contains a variable probability vector $\overrightarrow{P_j}$. Each $\overrightarrow{P_j}$ has n components, where n is the number of instructions in instruction set S. Each component $P_j(I)$ of $\overrightarrow{P_j}$ denotes the probability of choosing instruction $I \in S$ at node N_j. Each vector $\overrightarrow{P_j}$ is initialized as follows:

$$P_j(I) = \frac{P_T}{l} \quad \forall I : I \in T \tag{7}$$

$$P_j(I) = \frac{1 - P_T}{k} \quad \forall I : I \in F \tag{8}$$

PIPE combines two forms of learning: generation-based learning (GBL) and elitist learning (EL). GBL is PIPE's main learning algorithm. The purpose of EL is to use the best program found so far as an attractor. PIPE executes as follows:

```
GBL
REPEAT
   with probability $P_{el}$ DO EL
   otherwise DO GBL
UNTIL termination criterion is reached
```

Here P_{el} is a user-defined constant in [0, 1].

Generation-based learning

Step 1. *Creation of program population.* A population of programs P_{ROG_j} ($0 < j \leqslant \mathrm{PS}$; PS is population size) is generated using the prototype tree PPT.

Step 2. *Population evaluation.* Each program P_{ROG_j} of the current population is evaluated on the given task and assigned a fitness value $\mathrm{FIT}(P_{\mathrm{ROG}_j})$ according to the predefined fitness function (Eqs. (5) and (6)). The best program of the current population (the one with the smallest fitness value) is denoted P_{ROG_b}. The best program found so far (elitist) is preserved in $P^{\mathrm{el}}_{\mathrm{ROG}}$.

Step 3. *Learning from population.* Prototype tree probabilities are modified such that the probability $P(P_{\mathrm{ROG}_b})$ of creating P_{ROG_b} increases. This procedure is called adapting PPT towards (Progb). This is implemented as follows. First $P(P_{\mathrm{ROG}_b})$ is computed by looking at all PPT nodes N_j used to generate P_{ROG_b}:

$$P(P_{\mathrm{ROG}_b}) = \prod_{j:N_j \text{ used to generate } P_{\mathrm{ROG}_b}} P_j(I_j(P_{\mathrm{ROG}_b})) \tag{9}$$

where $I_j(P_{\mathrm{ROG}_b})$ denotes the instruction of program P_{ROG_b} at node position j. Then a target probability P_{TARGET} for P_{ROG_b} is calculated:

$$P_{\mathrm{TARGET}} = P(P_{\mathrm{ROG}_b}) + (1 - P(P_{\mathrm{ROG}_b})) \cdot \mathrm{lr} \cdot \frac{\varepsilon + \mathrm{FIT}(P^{\mathrm{el}}_{\mathrm{ROG}})}{\varepsilon + \mathrm{FIT}(P_{\mathrm{ROG}_b})} \tag{10}$$

Here 'lr' is a constant learning rate and ε a positive user-defined constant. Given P_{TARGET}, all single node probabilities $P_j(I_j(P_{\mathrm{ROG}_b}))$ are increased iteratively:

REPEAT:

$$P_j(I_j(P_{\mathrm{ROG}_b})) = P_j(I_j(P_{\mathrm{ROG}_b})) + c^{\mathrm{lr}} \cdot \mathrm{lr} \cdot (1 - P_j(I_j(P_{\mathrm{ROG}_b}))) \tag{11}$$

UNTIL

$$P(P_{\mathrm{ROG}_b}) \geqslant P_{\mathrm{TARGET}}$$

where c^{lr} is a constant influencing the number of iterations. The smaller c^{lr} the higher the approximation precision of P_{TARGET} and the number of required iterations. Setting $c^{\mathrm{lr}} = 0.1$ turned out to be a good compromise between precision and speed. And then all adapted vectors $\overrightarrow{P_j}$ are renormalized.

Step 4. *Mutation of prototype tree.* All probabilities $P_j(I)$ stored in nodes N_j that were accessed to generate program P_{ROG_b} are mutated with probability P_{M_p}:

$$P_{M_p} = \frac{P_{\mathrm{M}}}{n \cdot \sqrt{|P_{\mathrm{ROG}_b}|}} \tag{12}$$

where the user-defined parameter P_{M} defines the overall mutation probability, n is the number of instructions in instruction set S and $|P_{\mathrm{ROG}_b}|$ denotes the number of nodes in program P_{ROG_b}. Selected probability vector components are then mutated as follows:

$$P_j(I) = P_j(I) + \mathrm{mr} \cdot (1 - P_j(I)) \tag{13}$$

where 'mr' is the mutation rate, another user-defined parameter. Also all mutated vectors $\overrightarrow{P_j}$ are renormalized.

Step 5. *Prototype tree pruning.* At the end of each generation the prototype tree is pruned. PPT subtrees attached to nodes that contain at least one probability vector component above a threshold T_{P} can be pruned.

Step 6. *Termination criteria.* Repeat above procedure until a fixed number of program evaluations is reached or a satisfactory solution is found.

Elitist learning

Elitist learning focuses search on previously discovered promising parts of the search space. The PPT is adapted towards the elitist program $P^{\mathrm{el}}_{\mathrm{ROG}}$. This is realized by replacing the P_{ROG_b} with $P^{\mathrm{el}}_{\mathrm{ROG}}$ in learning from population in Step 3. It is particularly useful with small population sizes and works efficiently in the case of noise-free problems.

In order to learn the structure and parameters of a FNT simultaneously there is a tradeoff between the structure optimization and parameter learning.

In fact, if the structure of the evolved model is not appropriate, it is not useful to pay much attention to the parameter optimization. On the contrary, if the best structure has been found, further structure optimization may destroy the best structure. In this paper, a technique for balancing the structure optimization and parameter learning is proposed. If the better structure is found then do local search (simulated annealing) for a number of steps (maximum allowed steps) or stop in case no better parameter vector is found for a significantly long time (say 100–2000 in our experiments). The criterion of better structure is distinguished as follows: if the fitness value of the best program is smaller than the fitness value of the elitist program, or the fitness values of two programs are equal but the nodes of the former is lower than the later, then we say that the better structure is found.

3.2. Parameter optimization

To find the optimal parameters set (weights and activation function parameters) of a FNT model, a number of global and local search algorithms namely genetic algorithm, evolutionary programming, gradient based learning method etc. can be employed. A variant of simulated annealing (called degraded ceiling) is selected due to its straightforward property and fast local search capability [15].

Simulated annealing is one of the most widely studied local search metaheuristics. It was proposed as a general stochastic optimization technique in 1983 [16] and has been applied to solve a wide range of problems including connection weights optimization of a neural network.

The basic ideas of the simulated annealing search are that it accepts worse solutions with a probability $p = e^{-\frac{\delta}{T}}$, where $\delta = f(s^*) - f(s)$, the s and s^* are the old and new solution vectors, $f(s)$ denotes the cost function, the parameter T denotes the temperature in the process of annealing. Originally it was suggested to start the search from a high temperature and reduce it to the end of the process by a formula: $T_{i+1} = T_i - T_i * \beta$. However, the cooling rate β and initial value of T should be carefully selected due to it is problem dependent.

The degraded ceiling algorithm also keeps the acceptance of worse solutions but with a different manner. It accepts every solution whose objective function is less than or equal to the upper limit B, which is monotonically decreased during the search. The procedure of the degraded ceiling algorithm is given in Fig. 3.

3.3. The general learning algorithm

The general learning procedure for designing a FNT model may be described as follows.

```
Set the initial solution S
Calculate initial fitness function f(s)
Initial ceiling B=f(s)
Specify input parameter dB
While not some stopping condition do
   define neighbourhood N(s)
   Randomly select the candidate solution s* in N(s)
   If ( f(s*) < f(s) )   or  ( f(s*) <= B )
   Then accept s*
```

Fig. 3. The Degraded ceiling algorithm.

1. Set the initial values of parameters used in the PIPE and SA algorithms. Set the elitist program as NULL and its fitness value as a biggest positive real number of the computer at hand. Create the initial population (flexible neural trees and their corresponding parameters).
2. Structure optimization by PIPE algorithm as described in Section 3.1 in which the fitness function is calculated by mean square error (MSE) or root mean square error (RMSE).
3. If the better structure is found, then go to step 4, otherwise go to step 2.
4. Parameter optimization is achieved by the degraded ceiling algorithm as described in Section 3.2. In this stage, the tree structure or architecture of flexible neural tree model is fixed, and the best tree is taken from the end of run of the PIPE search. All the parameters used in the best tree formulated a parameter vector to be optimized by local search.
5. If the maximum number of iterations of SA algorithm is reached, or no better parameter vector is found for a significantly long time (100 steps) then go to step 6; otherwise go to step 4.
6. If satisfactory solution is found, then stop; otherwise go to step 2.

4. Experimental results and illustrative examples

The developed flexible neural tree model is applied here in conjunction with two time-series prediction problems: Box–Jenkins and Mackey-Glass chaotic time series. Well-known benchmark examples are used for the sake of easy comparison with existing models. The data related to the examples are available on the web site of the Working Group on Data Modeling Benchmark—IEEE Neural Network Council [17].

For each benchmark problem, two experimental simulations are carried out. The first one use the same inputs with other models so as to make a meaningful comparison. The second one use a large number of input variables in order the

Table 1
Parameters used in the flexible neural tree model

Parameter	Initial value
Population size, PS	30
Elitist learning probability, P_{el}	0.01
Learning rate, lr	0.01
Fitness constant, ε	0.000001
Overall mutation probability, P_M	0.4
Mutation rate, mr	0.4
Prune threshold, T_P	0.999999
Maximum local search steps	2000
Initial connection weights	rand[−1, 1]
Initial parameters, a_i and b_i	rand[0, 1]

FNT to select proper input variables or time-lags automatically. In addition, the parameters used for each experiment is listed in Table 1.

4.1. Application to Jenkins–Box time series

The gas furnace data (series J) of Box and Jenkins (1970) was recorded from a combustion process of a methane–air mixture. It is well known and frequently used as a benchmark example for testing identification and prediction algorithms. The data set consists of 296 pairs of input-output measurements. The input $u(t)$ is the gas flow into the furnace and the output $y(t)$ is the CO_2 concentration in outlet gas. The sampling interval is 9 s.

4.1.1. Case 1

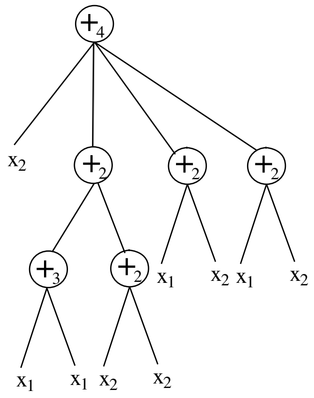
The inputs for constructing FNT model are $u(t-4)$ and $y(t-1)$, and the output is $y(t)$.

In this study, 200 data samples are used for training and the remaining data samples are used for testing the performance of the evolved model. The used instruction set for creating a FNT model is $S = F \cup T = \{+_2, +_3, \ldots, +_8\} \cup \{x_1, x_2\}$. Where x_1 and x_2 denotes the input variables $u(t-4)$ and $y(t-1)$, respectively.

After 37 generations, the optimal neural tree model was obtained with the MSE 0.000664. The MSE value for validation data set is 0.000701. The evolved neural tree is shown in Fig. 4 (left) and the actual time-series, the FNT model output and the prediction error is shown in Fig. 4 (right).

4.1.2. Case 2

For the second simulation, 10 inputs variables are used for constructing a FNT model. The proper time-lags for constructing a FNT model are finally determined by an evolutionary procedure.



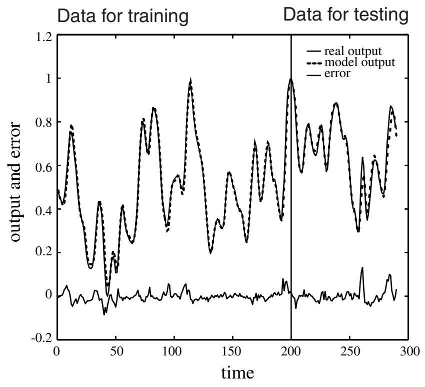
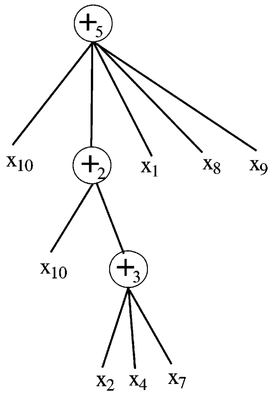


Fig. 4. Case 1: The evolved FNT model for prediction of Jenkins–Box data (left), and the actual time-series data, output of the evolved FNT model and the prediction error for training and test samples (right).

The used instruction sets to create an optimal neural tree model is $S = F \cup T = \{+_2, \ldots, +_8\} \cup \{x_1, x_2, \ldots, x_{10}\}$, where x_i $(i = 1, 2, \ldots, 10)$ denotes $u(t-6)$, $u(t-5)$, $u(t-4)$, $u(t-3)$, $u(t-2)$, $u(t-1)$ and $y(t-1)$, $y(t-2)$, $y(t-3)$, $y(t-4)$, respectively.

After 17 generations of the evolution, the optimal neural tree model was obtained with MSE 0.000291. The MSE value for validation data set is 0.000305.



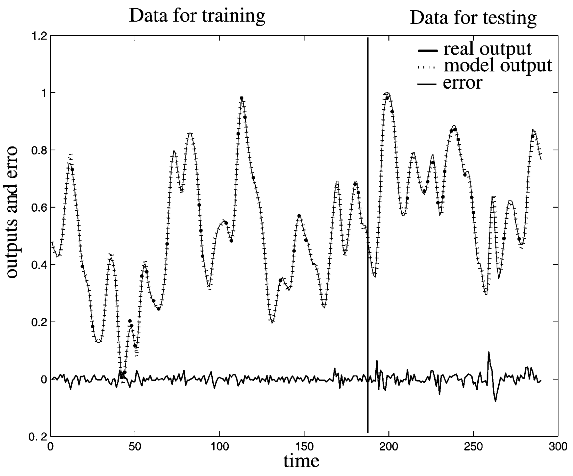


Fig. 5. Case 2: The evolved neural tree model for prediction of Jenkins–Box data (left), and the actual time series data, output of the evolved neural tree model and the prediction error for training and test samples (right).

Table 2
Comparison of prediction errors using different methods for the gas furnace data

Model name and reference	Number of inputs	MSE
ARMA [18]	5	0.71
Tong's model [19]	2	0.469
Pedrycz's model [20]	2	0.320
Xu's model [21]	2	0.328
Sugeno's model [22]	2	0.355
Surmann's model [23]	2	0.160
TS model [24]	6	0.068
Lee's model [25]	2	0.407
Hauptmann's model [26]	2	0.134
Lin's model [27]	5	0.261
Nie's model [28]	4	0.169
ANFIS model [29]	2	0.0073
FuNN model [30]	2	0.0051
HyFIS model [31]	2	0.0042
FNT model (Case 1)	2	0.00066
FNT model (Case 2)	7	0.00029

The evolved FNT is shown in Fig. 5 (left) and the actual time-series, the FNT model output and the prediction error is shown in Fig. 5 (right). From the evolved FNT tree, it can be seen that the optimal inputs variables for constructing a FNT model are: $u(t-6)$, $u(t-5)$, $u(t-3)$, $y(t-1)$, $y(t-2)$, $y(t-3)$ and $y(t-4)$. It should be noted that the FNT model with proper selected input variables has accurate precision and good generalization ability. A comparison result of different methods for forecasting Jenkins–Box data is shown in Table 2.

4.2. Application to Mackey-Glass time-series

The chaotic Mackey-Glass differential delay equation is recognized as a benchmark problem that has been used and reported by a number of researchers for comparing the learning and generalization ability of different models. The Mackey-Glass chaotic time series is generated from the following equation:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{14}$$

where $\tau > 17$, the equation shows chaotic behavior.

4.2.1. Case 1

To make the comparison with earlier work fair, we predict the $x(t+6)$ with using the inputs variables $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$. 1000 sample points used in our study. The first 500 data pairs of the series were used as training data, while the remaining 500 were used to validate the model identified.

The used instruction sets to create an optimal FNT model is $S = F \cup T = \{+_5, \ldots, +_{10}\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i ($i = 1, 2, 3, 4$) denotes $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$, respectively.

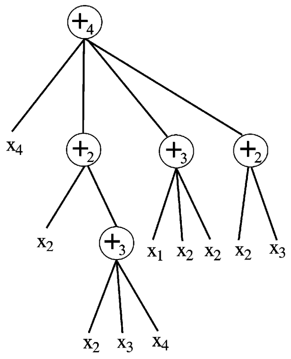
After 135 generations of the evolution, an optimal FNT model was obtained with RMSE 0.006901. The RMSE value for validation data set is 0.007123. The evolved FNT is shown in Fig. 6 (left). The actual time-series data, the output of FNT model and the prediction error are shown in Fig. 6 (right). A comparison result of different methods for forecasting Mackey-Glass data is shown in Table 3.

4.2.2. Case 2

For the second simulation, 19 inputs variables are used for constructing a FNT model. The proper time-lags for constructing a FNT model are finally determined by an evolutionary procedure.

The used instruction sets to create an optimal neural tree model is $S = F \cup T = \{+_2, \ldots, +_8\} \cup \{x_1, x_2, \ldots, x_{19}\}$, where $x_i (i = 1, 2, \ldots, 19)$ denotes $x(t-18)$, $x(t-17)$, ..., $x(t-1)$ and $x(t)$, respectively.

The optimal neural tree model was obtained with RMSE 0.00271. The RMSE value for validation data set is 0.00276. The evolved FNT is shown in Fig. 7 (left) and the actual time-series, the FNT model output and the prediction error is shown in Fig. 7 (right). From the evolved FNT, it can be seen



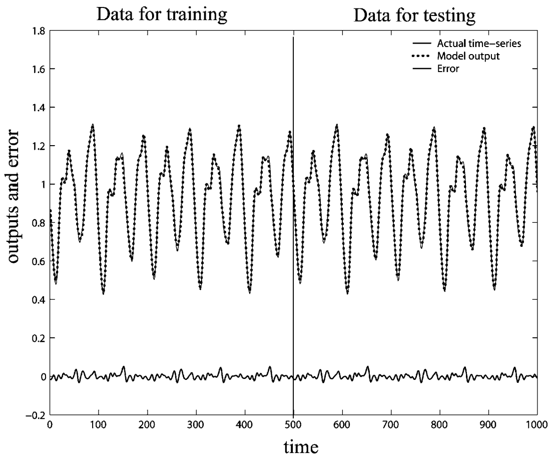
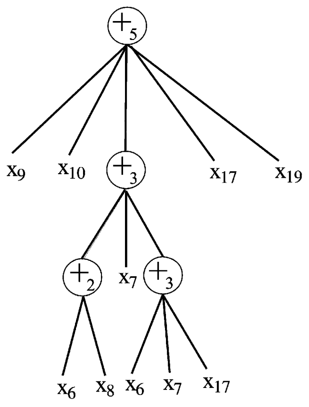


Fig. 6. Case 1: The evolved neural tree model for prediction of the Mackey-Glass time-series (left), and the actual time series data, output of the evolved neural tree model and the prediction error (right).

Table 3
Comparison of prediction error using different methods for the Mackey-Glass time-series problem

Method	Prediction error (RMSE)
Autoregressive model	0.19
Cascade correlation NN	0.06
Back-propagation NN	0.02
Sixth-order polynomial	0.04
Linear prediction method	0.55
ANFIS and Fuzzy System [29]	0.007
Wang et al. [33] Product T-norm	0.0907
Classical RBF (with 23 neurons) [32]	0.0114
PG-RBF network [34]	0.0028
Genetic algorithm and fuzzy system [35]	0.049
FNT model (Case 1)	0.0069
FNT model (Case 2)	0.0027



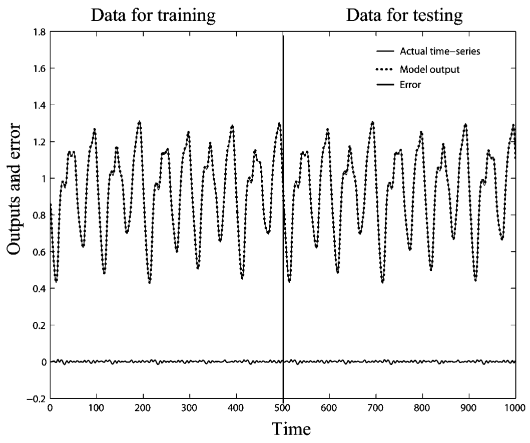


Fig. 7. Case 2: The evolved neural tree model for prediction of the Mackey-Glass time-series (left), and the actual time series data, output of the evolved neural tree model and the prediction error (right).

that the optimal inputs variables for constructing a FNT model are: $x(t-13)$, $x(t-12)$, $x(t-11)$, $x(t-10)$, $x(t-9)$, $x(t-2)$ and $x(t)$. That is, for the prediction of $x(t+6)$, among the time-lags from 0 to 18, the automatically evolved time-lags are 13, 12, 11, 10, 9, 2 and 0. It should be noted that the FNT model with proper selected time-lags as input variables has accurate precision and good generalization ability. A comparison result of different methods for forecasting Mackey-Glass data is shown in Table 3.

From the above simulation results, it can be seen that the proposed FNT model works well for generating prediction models of time series.

5. Concluding remarks

A new time-series forecasting model based on flexible neural tree is proposed in this paper. From the architecture perspective, a FNT can be seen as a flexible multi-layer feedforward neural network with over-layer connections and free parameters in activation functions. The work demonstrates that the FNT model with automatically selected input variables (time-lags) has better accuracy (low error) and good generalization ability. Simulation results for the time-series forecasting problems shown the feasibility and effectiveness of the proposed method.

Acknowledgment

This research was partially supported by the National Natural Science Foundation of China (NSFC), Project No. 69902005 and Provincial Natural Science Foundation of Shandong, Project No. Y2001G09.

References

[1] S. Omatu, Marzuki Khalid, Rubiyah Yusof, Neuro-Control and its Applications, Springer Publisher, 1996.
[2] S.E. Fahlman, Christian Lebiere, The cascade-correlation learning architecture, Advances in Neural Information Processing Systems 2 (1990) 524–532.
[3] J.-P. Nadal, Study of a growth algorithm for a feedforward network, International Journal of Neural Systems 1 (1989) 55–59.
[4] R. Setiono, L.C.K. Hui, Use of a quasi-Newton method in a feedforward neural network construction algorithm, IEEE Transactions on Neural Networks 6 (1995) 273–277.
[5] P.J. Angeline, Gregory M. Saunders, Jordan B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks 5 (1994) 54–65.
[6] X. Yao, Y. Liu, A new evolutionary system for evolving artificial neural networks, IEEE Transactions on Neural Networks 8 (1997) 694–713.
[7] X. Yao, Evolving artificial neural networks, Proceedings of the IEEE 87 (1999) 1423–1447.
[8] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (1999) 82–102.
[9] Kenneth O. Stanley, Risto Miikkulainen, Evolving neural networks through augmenting topologies, Evolutionary Computation 10 (2002) 99–127.
[10] B.T. Zhang, P. Ohm, H. Muhlenbein, Evolutionary induction of sparse neural trees, Evolutionary Computation 5 (1997) 213–236.
[11] G. Peter Zhang, Time series forecasting using a hybrid ARIMA and neural network model, Neurocomputing 50 (2003) 159–175.
[12] R.P. Salustowicz, J. Schmidhuber, Probabilistic Incremental Program Evolution, Evolutionary Computation 2 (5) (1997) 123–141.
[13] Y. Chen, S. Kawaji, System identification and control using probabilistic incremental program evolution algorithm, Journal of Robotics and Machatronics 12 (2000) 675–681.
[14] S. Baluja, Population-based incremental learning: a method for integrating genetic search based function optimization and competitive learning, Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, 1994.

[15] E.K. Burke, Y. Bykov, J.P. Newall, S. Petrovic, A new local search approach with execution time as an input parameter, Technical Report No. NOTTCS-TR-2002-3, School of Computer Science and Information Technology, University of Nottingham, 2002.
[16] S. Kirkpatrick Jr., C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[17] Working Group on Data Modeling Benchmark, Standard Committee of IEEE Neural Network Council. <http://neural.cs.nthu.edu.tw/jang/benchmark/>, 2004 (accessed 14.10.04).
[18] G.E.P. Box, Time Series Analysis, Forecasting and Control, Holden Day, San Francisco, 1970.
[19] R.M. Tong, The evaluation of fuzzy models derived from experimental data, Fuzzy Sets and Systems 4 (1980) 1–12.
[20] W. Pedtycz, An identification algorithm in fuzzy relational systems, Fuzzy Sets and Systems 13 (1984) 153–167.
[21] C.W. Xu, Y.Z. Lu, Fuzzy model identification and self-learning for dynamic systems, IEEE Transactions on Systems, Man and Cybernetics 17 (1987) 683–689.
[22] M. Sugeno, T. Takagi, Linguistic modelling based on numerical data, Proceedings of the IFSA'91, 1991.
[23] H. Surmann, A. Kanstein, K. Goser, Self-organising and genetic algorithm for an automatic design of fuzzy control and decision systems, Proceedings of the FUFIT's93 (1993) 1079–1104.
[24] M. Sugeno, T. Takagi, A fuzzy-logic approach to qualitative modeling, IEEE Transactions on Fuzzy Systems 1 (1993) 7–31.
[25] Y.-C. Lee, E. Hwang, Y.-P. Shih, A combined approach to fuzzy model identification, IEEE Transactions on Systems, Man and Cybernetics 24 (1994) 736–744.
[26] W. Hauptmann, A neural net topology for bidirectional fuzzy-neuro transformation, Proceedings of the IEEE International Conference on Fuzzy Systems (1995) 1511–1518.
[27] Y. Lin, G.A. Cunningham, A new approach to fuzzy-neural system modelling, IEEE Transactions on Fuzzy Systems 3 (1995) 190–197.
[28] J. Nie, Constructing fuzzy model by self-organising counter propagation network, IEEE Transactions on Systems Man and Cybernetics 25 (1995) 963–970.
[29] J.-S.R. Jang, C.-T. Sun, E. Mizutani, Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence, Prentice-Hall, Upper Saddle River, NJ, 1997.
[30] N. Kasabov, J.S. Kim, M. Watts, A. Gray, FuNN/2—a fuzzy neural network architecture for adaptive learning and knowledge acquisition, Information Science 101 (1996) 155–175.
[31] J. Kim, N. Kasabov, HyFIS: adaptive neuro-fuzzy inference systems and their application to nonlinear dynamical systems, Neural Networks 12 (1999) 1301–1319.
[32] K.B. Cho, B.H. Wang, Radial basis function based adaptive fuzzy systems their application to system identification and prediction, Fuzzy Sets and Systems 83 (1995) 325–339.
[33] L.X. Wang, J.M. Mendel, Generating fuzzy rules by learning from examples, IEEE Transactions on Systems, Man and Cybernetics 22 (1992) 1414–1427.
[34] I. Rojas, H. Pomares, J. Luis Bernier et al., Time series analysis using normalized PG-RBF network with regression weights, Neurocomputing 42 (2002) 267–285.
[35] D. Kim, C. Kim, Forecasting time series with genetic fuzzy predictor ensembles, IEEE Transactions on Fuzzy Systems 5 (1997) 523–535.
[36] X. Li, W. Yu, Dynamic system identification via recurrent multilayer perceptions, Information Science 147 (2002) 45–63.
[37] J.-H. Horng, Neural adaptive tracking control of a DC motor, Information Sciences 118 (1999) 1–13.
[38] H. Kirschner, R. Hillebr, Neural networks for HREM image analysis, Information Sciences 129 (2000) 31–44.
[39] A.F. Sheta, K.D. Jong, Time-series forecasting using GA-tuned radial basis functions, Information Science 133 (2001) 221–228.

[40] L. Snchez, I. Cousob, J.A. Corrales, Combining GP operators with SA search to evolve fuzzy rule based classifiers, Information Sciences 136 (2001) 175–191.

[41] Y.S. Yeun, J.C. Suh, Y.S. Yang, Function approximations by superimposing genetic programming trees: with applications to engineering problems, Information Sciences 122 (2000) 259–280.