A Design of EA-based Self-Organizing Polynomial Neural Networks using Evolutionary Algorithm for Nonlinear System Modeling

Dong-Won Kim and Gwi-Tae Park

Department of Electrical Engineering, Korea University,

1, 5-ka, Anam-dong, Seongbuk-ku, Seoul 136-701, Korea.

Tel: +82-2-929-5185, Fax: +82-2-929-5185

E-mail: dwkim@elec.korea.ac.kr

Abstract: We discuss a new design methodology of self-organizing approximator technique (self-organizing polynomial neural networks (SOPNN)) using evolutionary algorithm (EA). The SOPNN dwells on the ideas of group method of data handling. The performances of SOPNN depend strongly on the number of input variables available to the model, the number of input variables and type (order) of the polynomials to each node. They must be fixed by designer in advance before the architecture is constructed. So the trial and error method must go with heavy computation burden and low efficiency. Moreover it does not guarantee that the obtained SOPNN is the best one. In this paper, we propose EA-based SOPNN to alleviate these problems. The order of the polynomial, the number of input variables, and the optimum input variables are encoded as a chromosome and fitness of each chromosome is computed. So the appropriate information of each node is evolved accordingly and tuned gradually throughout the EA iterations. We can show that the EA-based SOPNN is a sophisticated and versatile architecture which can construct models for limited data set as well as poorly defined complex problems. Comprehensive comparisons show that the performance of the EA-based SOPNN is significantly improved in the sense of approximation and prediction abilities with a much simpler structure compared with the conventional SOPNN model as well as previous identification methods.

Keywords: SOPNN, evolutionary algorithm, approximation and prediction abilities, identification methods.

1. Introduction

System modeling and identification is important for system analysis, control, and automation as well as for scientific research. So a lot of attention has been directed to developing advanced techniques of system modeling. Neural networks and fuzzy systems have been widely used for modeling nonlinear systems. The approximation capability of neural networks, such as multilayer perceptrons, radial basis function (RBF) networks, or dynamic recurrent neural networks has been investigated by many authors [1-3]. On the other hand, fuzzy systems have been proved to be able to approximate nonlinear functions with arbitrary accuracy [4-5]. But the resultant neural network representation is very complex and difficult to understand and fuzzy systems require too many fuzzy rules for accurate function approximation, particularly in the case of multidimensional input. As another method, there is a GMDH-type algorithm. Group Method of Data Handling (GMDH) was introduced by Ivakhnenko in the early 1970's [6-10]. GMDH-type algorithms have been extensively used since the mid-1970's for prediction and modeling complex nonlinear processes. The main characteristics of GMDH is that it is a self-organizing and provides an automated selection of essential input variables without using a prior information on the relationship among input-output variables [11]. Self-organizing Polynomial Neural Networks (SOPNN) [12-13] is GMDH-type algorithm and one of useful approximator techniques. SOPNN has an architecture similar to feedforward neural networks whose neurons are replaced by polynomial nodes. The output of the each node in SOPNN structure is obtained using several types of high-order polynomial such as linear, quadratic, and modified quadratic of input variables. These polynomials are called as partial descriptions (PDs). SOPNNs have fewer nodes than Artificial Neural Networks (ANNs), but the nodes are more flexible. The SOPNN shows a superb performance in comparison to the previous fuzzy modeling methods. Although the SOPNN is structured by a systematic design procedure, it has some drawbacks to be solved. If there are sufficiently large

number of input variables and data points, SOPNN algorithm has a tendency to produce overly complex networks. On the other hand, if a small number of input variables are available, SOPNN does not maintain good performance. Moreover, the performances of SOPNN depend strongly on the number of input variables available to the model, the number of input variables and types or order in each PD. They must be chosen in advance before the architecture of SOPNN is constructed. In most cases, they are determined by the trial and error method with a heavy computational burden and low efficiency. Moreover, the SOPNN algorithm is a heuristic method so it does not guarantee that the obtained SOPNN is the best one for nonlinear system modeling. Therefore, more attention must be paid to solve the above-mentioned drawbacks.

In this paper we will present a new design methodology of SOPNN using evolutionary algorithm (EA) in order to alleviate the above-mentioned drawbacks of the SOPNN. We call this new network the EA-based SOPNN.

Evolutionary Algorithm (EA) has been widely used as a parallel global search method for optimization problems [14-17]. The EA is used to determine that how many input variables are chosen to each node, which input variables are optimally chosen among many input variables, and what is the appropriate type of the polynomials in each PD.

This paper is organized as follows. The design procedure of the conventional SOPNN is briefly described in Section 2. A design methodology of EA-based SOPNN is described in Section 3. Coding of the key factors of the SOPNN, the representation of chromosome and fitness function are also discussed in Section 3. The proposed EA-based SOPNN is applied to nonlinear systems modeling to show its performances compared with other methods including conventional SOPNN in Section 4, Finally conclusions are given in Section 5.

2. Design procedure of SOPNN

The SOPNN algorithm is based on the GMDH method and utilizes a class of polynomials such as linear, quadratic, and modified quadratic types. By choosing the most significant input variables and polynomial types among various types of forms available, we can obtain the PDs in each layer. The framework of the design procedure of the SOPNN comes as a sequence of the following steps.

[Step 1] *Determine system's input variables.*

We define the input variables such as $x_{1i}, x_{2i}, \cdots x_{Ni}$ related to output variables y_i, where N and i are the number of entire input variables and input-output data set, respectively. The normalization of the input data is also performed if required.

[Step 2] *Form training and testing data.*

The input - output data set is separated into training (n_{tr}) data set and testing (n_{te}) data set. Obviously we have $n = n_{tr} + n_{te}$. The training data set is used to construct a SOPNN model. And the testing data set is used to evaluate the constructed SOPNN model.

[Step 3] *Choose a structure of the SOPNN.*

The structure of SOPNN is strongly dependent on the number of input variables and the order of PD in each layer. Two kinds of SOPNN structures, namely, the basic SOPNN structure and the modified SOPNN structure can be available. Each of them comes with two cases. Table 1 summarizes the various SOPNN structures.

(a) Basic SOPNN structure – The number of input variables of PDs is the same in every layer.

Case 1. The polynomial order of the PDs is the same in each layer of the network.

Case 2. The polynomial order of the PDs in the 2nd or higher layer is different from the one of PDs in the 1st layer.

(b) Modified SOPNN structure – The number of input variables of PDs varies from layer to layer.

Case 1. The polynomial order of the PDs is same in every layer.

Case 2. The polynomial order of the PDs in the 2nd layer or higher is different from the one of PDs

in the 1st layer.

Table 1. Taxonomy of various SOPNN structures

PD Type / Layer	No. of input variables	Order of Polynomial	SOPNN structure
1st layer	i	Type I	(1) i=j: Basic SOPNN a) I=J: Case 1 b) I≠J: Case 2 (2) i≠j: Modified SOPNN a) I=J: Case 1 b) I≠J: Case 2
2-5th layer	j	Type J	

(i, j=2, 3, …, ; I, J=1, 2, 3)

[Step 4] *Determine the number of input variables and the order of the polynomial forming a PD.*

We determine arbitrarily the number of input variables and the type of the polynomial in PDs. The polynomials are different according to the number of input variables and the polynomial order. Several types of polynomials are shown in the Table 2. The total number of PDs located at the current layer is determined by the number of the selected input variables (r) from the nodes of the preceding layer, because the outputs of the nodes of the preceding layer become the input variables to the current layer. The total number of PDs in the current layer is equal to the combination ${}_N c_r$, that is $\frac{N!}{r!(N-r)!}$, where N is the number of nodes in the preceding layer.

Table 2. Different types of the polynomial in PDs.

No. of inputs / Order of the polynomial	1	2	3
1 (Type 1)	Linear	Bilinear	Trilinear
2 (Type 2)	Quadratic	Biquadratic	Triquadratic
2 (Type 3)	Modified quadratic	Modified biquadratic	Modified triquadratic

- Bilinear PD = $c_0 + c_1 x_1 + c_2 x_2$
- Biquadratic PD = $c_0 + c_1 x_1 + c_2 x_2 + c_3 x_1^2 + c_4 x_2^2 + c_5 x_1 x_2$

- Modified biquadratic PD = $c_0 + c_1x_1 + c_2x_2 + c_3x_1x_2$
- Trilinear PD = $c_0 + c_1x_1 + c_2x_2 + c_3x_3$
- Triquadratic PD= $c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_1^2 + c_5x_2^2 + c_6x_3^2 + c_7x_1x_2 + c_8x_1x_3 + c_9x_2x_3$
- Modified triquadratic PD = $c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_1x_2 + c_5x_1x_3 + c_6x_2x_3$

[Step 5] *Estimate the coefficients of the PD.*

The vector of coefficients of the PDs as shown in Table 2 is determined using a standard mean squared errors (MSE) by minimizing the following index

$$E_k = \frac{1}{n_{tr}} \sum_{i=1}^{n_{tr}} (y_i - z_{ki})^2, \quad k = 1,2,\cdots \frac{N!}{r!(N-r)!} \tag{1}$$

where, z_{ki} denotes the output of the *k*-th node with respect to the *i*-th data and n_{tr} is the number of training data subset.

This step is completed repeatedly for all the nodes in the current layer and, in the sequel, all layers of the SOPNN starting from the input to the output layer.

[Step 6] *Select PDs with the good predictive capability.*

The predictive capability of each PD is evaluated by performance index using the testing data set. Then we choose *w* PDs Among ${}_N c_r$ PDs in due order from the best predictive capability (the lowest value of the performance index). Here, *w* is the pre-defined number of PDs that must be preserved to next layer. The outputs of the chosen PDs serve as inputs to the next layer.

There are two cases as to the number of the preserved PDs in each layer

If $\frac{N!}{r!(N-r)!} < w$ then the number of the chosen PDs retained for the next layer is equal to $\frac{N!}{r!(N-r)!}$

If $\frac{N!}{r!(N-r)!} \geq w$ then the number of the chosen PDs retained for the next layer is equal to *w*

[Step 7] *Check the stopping criterion.*

The SOPNN algorithm terminates when the number of layers predetermined by the designer is reached.

[Step 8] *Determine new input variables for the next layer.*

If the stopping criterion is not satisfied, the next layer is constructed by repeating step 4 through step 8.

The overall architecture of the SOPNN is shown in Fig. 1. When the final layer has been constructed, the node with the best predictive capability is selected as the output node. All remaining nodes except output node in the final layer are discarded. Furthermore, all the nodes in the previous layers that do not have influence on the output node are also removed by tracing the data flow path of each layer.

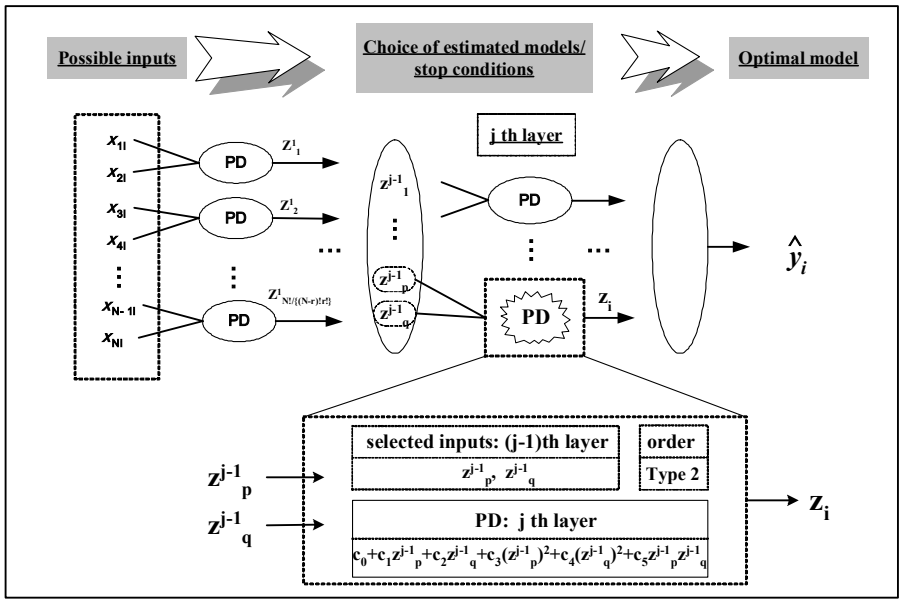


Fig. 1. Overall architecture of the SOPNN

The SOPNN is a flexible neural architecture whose structure is developed through modeling process. In particular, the number of the layers and the number of nodes in each layer of the SOPNN are not fixed in advance (it usually happens in the case of multilayer perceptron) but generated in a dynamic way. Its each node exhibits a high level of flexibility and realizes a polynomial type of mapping between input and output variables. As a result, SOPNN provides a systematic design procedure but the performances depend strongly on a few factors stated in the section 1. In the following section, we propose the new design procedure using EA for the systemic design of SOPNN with the optimum performance.

3. Design of EA-based SOPNN

In this section, a new design technique of SOPNN using EA is described. In the SOPNN algorithm, the problems are how to determine the optimal number of input variables, which input variables are chosen, and how to select the order of the polynomial forming a PD in each node. In this paper, these problems are solved by using EA. The EA is implemented using crossover and mutation probably rates for better exploitation of the optimal inputs and order of polynomial in each node of SOPNN. All of the initial EA populations are randomized, which implies that minimum heuristic knowledge is used. The appropriate inputs and order are evolved accordingly and are tuned gradually throughout the EA iterations.

In the evolutionary design procedure, key issues are how to encode the order of the polynomial, the number of input variables, and the optimum input variables as a chromosome and how to define a criterion to compute the fitness of each chromosome. In what follows, the detailed representation of the coding strategy and choice of fitness function are given.

3.1 Representation of chromosome for appropriate information of each PD

When we design the SOPNN using EA, the most important consideration is the representation strategy, that is how to encode the key factors of the SOPNN into the chromosome. We employ a binary coding for the available design specification. We code the order and the inputs of each node in the SOPNN as a finite-length string. Our chromosomes are made of three sub-chromosomes. The first one is consisted of 2 bits for the order of polynomial (PD), the second one is consisted of 3 bits for the number of inputs of PD, and the last one is consisted of N bits which are equal to the number of entire input candidates in the current layer. These input candidates are the node outputs of the previous layer. The representation of binary chromosomes is illustrated in Fig. 2.

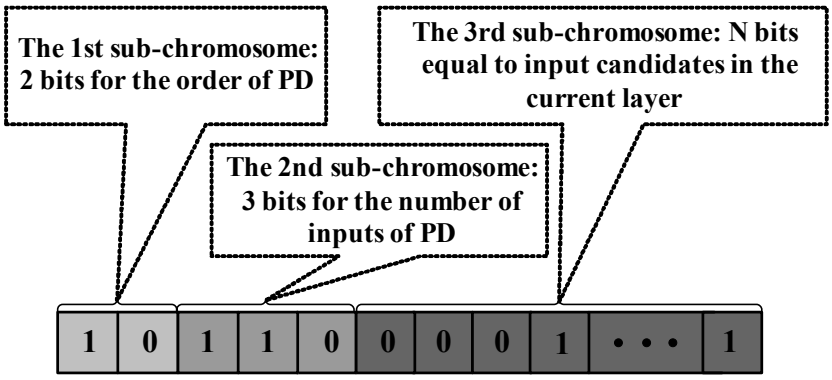


Fig. 2. Structure of binary chromosome for a PD

The 1st sub-chromosome is made of 2 bits. It represents several types of order of PD. The relationship between bits in the 1st sub-chromosome and the order of PD is shown in Table 3. Thus, each node can exploit a different order of the polynomial.

Table 3. Relationship between bits in the 1st sub-chromosome and order of PD.

Bits in the 1st sub-chromosome	Order of polynomial(PD)
00	Type 1 – Linear
01	Type 2 – Quadratic
10	
11	Type 3 – Modified quadratic

The 3rd sub-chromosome has N bits, which are concatenated a bit of 0's and 1's coding. The input candidate is represented by a 1 bit if it is chosen as input variable to the PD and by a 0 bit it is not chosen. This way solves the problem of which input variables to be chosen.

If many input candidates are chosen for model design, the modeling is computationally complex, and normally requires a lot of time to achieve good results. In addition, it causes improper results and poor generalization ability. Good approximation performance does not necessarily guarantee good generalization capability [18]. To overcome this drawback, we introduce the 2nd sub-chromosome into the chromosome. The 2nd sub-chromosome is consisted of 3 bits and represents the number of input variables to be selected. The number based on the 2nd sub-chromosome is shown in the Table 4. Input variables for

each node are selected among entire input candidates as many as the number represented in the 2nd sub-chromosome. Designer must determine the maximum number in consideration of the characteristic of system, design specification, and some prior knowledge of model. With this method we can solve the problems such as the conflict between overfitting and generalization and the requirement of a lot of computing time.

Table 4. Relationship between bits in the 2nd sub-chromosome and number of inputs to PD.

Bits in the 2nd sub-chromosome	Number of inputs to PD
000	1
001	2
010	2
011	3
100	3
101	4
110	4
111	5

The relationship between chromosome and information on PD is shown in Fig. 3. The PD corresponding to the chromosome in Fig. 3 is described briefly as Fig. 4.

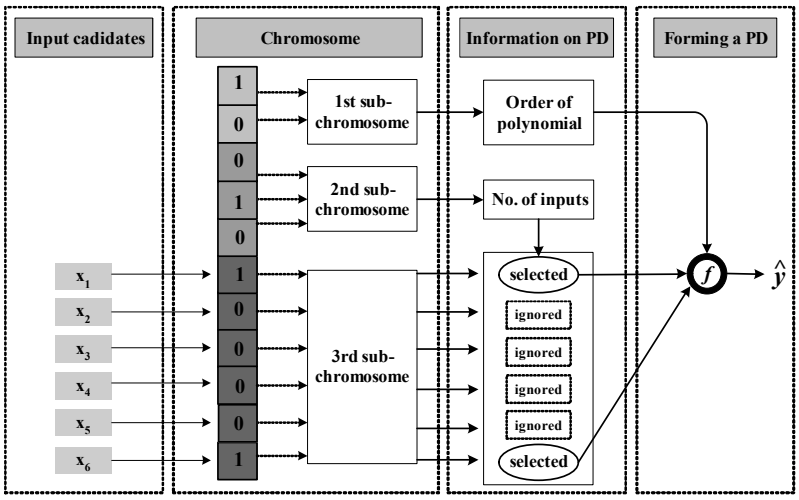


Fig. 3. Example of PD whose various pieces of required information are obtained from its chromosome

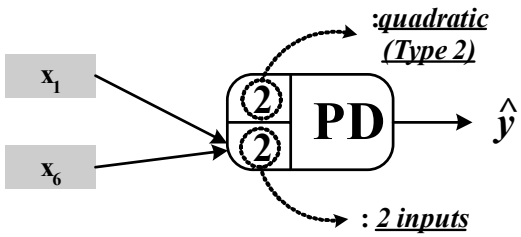


Fig. 4. Node with PD corresponding to chromosome in Fig. 3.

Fig. 3 shows an example of PD. The various pieces of required information are obtained its chromosome. The 1st sub-chromosome shows that the polynomial order is Type 2 (quadratic form). The 2nd sub-chromosome shows two input variables to this node. The 3rd sub-chromosome tells that x_1 and x_6 are selected as input variables. The node with PD corresponding to Fig. 3 is shown in Fig. 4. Thus, the output of this PD $\hat{y}$ can be expressed as (2).

$$\hat{y} = f(x_1, x_6) = c_0 + c_1 x_1 + c_2 x_6 + c_3 x_1^2 + c_4 x_6^2 + c_5 x_1 x_6 \tag{2}$$

where coefficients c_0, c_1, …, c_5 are evaluated using the training data set by means of the standard LSE.

The polynomial function, PD, is formed automatically according to the information of sub-chromosomes.

The design procedure of EA-based SOPNN is shown in Fig. 5. At the beginning of the process, the initial populations comprise a set of chromosomes that are scattered all over the search space. The populations are all randomly initialized. Thus, the use of heuristic knowledge is minimized. The assignment of the fitness in EA serves as guidance to lead the search toward the optimal solution. Fitness function with several specific cases for modeling will be explained later. After each of the chromosomes is evaluated and associated with a fitness, the current population undergoes the reproduction process to create the next generation of population. The roulette-wheel selection scheme is used to determine the members of the new generation of population. After the new group of population is built, the mating pool is formed and the crossover is carried out. The crossover proceeds in three steps. First, two newly reproduced strings are selected from the mating pool produced by reproduction. Second, a position (one point) along the two

strings is selected uniformly at random. The third step is to exchange all characters following the crossing site. We use one-point crossover operator with a crossover probability of P_c (0.85). This is then followed by the mutation operation. The mutation is the occasional alteration of a value at a particular bit position (we flip the states of a bit from 0 to 1 or vice versa). The mutation serves as an insurance policy which would recover the loss of a particular piece of information (any simple bit). The mutation rate used is fixed at 0.05 (P_m). Generally, after these three operations, the overall fitness of the population improves. Each of the population generated then goes through a series of evaluation, reproduction, crossover, and mutation, and the procedure is repeated until a termination condition is reached. After the evolution process, the final generation of population consists of highly fit bits that provide optimal solutions. After the termination condition is satisfied, one chromosome (PD) with the best performance in the final generation of population is selected as the output PD. All remaining other chromosomes are discarded and all the nodes that do not have influence on this output PD in the previous layers are also removed. By doing this, the EA-based SOPNN model is obtained.

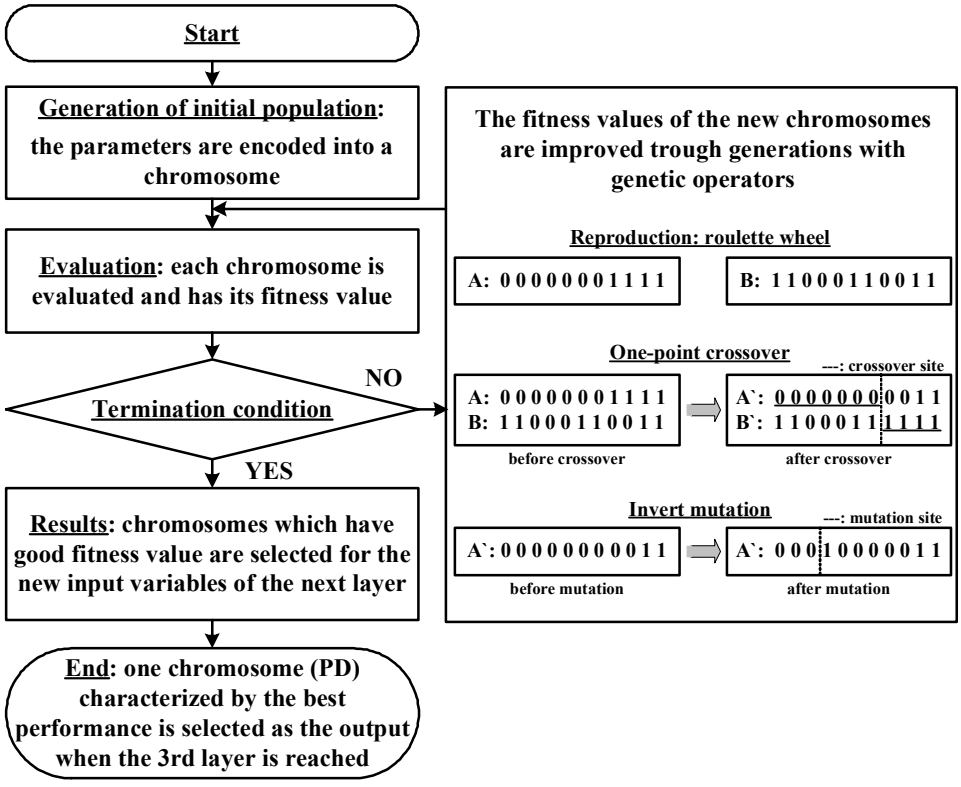


Fig. 5. Block diagram of the design procedure of EA-based SOPNN.

3.2 Fitness function for modeling

The important thing to be considered for the EA is the determination of the fitness function. The genotype representation encodes the problem into a string while the fitness function measures the performance of the model. It is quite important for evolving systems to find a good fitness measurement. To construct models with significant approximation and generalization ability, we introduce the error function such as

$$E = \theta \times PI + (1-\theta) \times EPI \tag{3}$$

where $\theta \in [0,1]$ is a weighting factor for PI and EPI, which denote the values of the performance index for the training data and testing data, respectively. Then the fitness value is determined as follows:

$$F = \frac{1}{1+E} \tag{4}$$

Maximizing F is identical to minimizing E. The choice of θ establishes a certain tradeoff between the approximation and generalization ability of the EA-based SOPNN.

4. Simulation results

In this section, we show the performance of our new EA-based SOPNN for two well known nonlinear system modeling. One is a time series of gas furnace (Box-Jenkins data)[19] which was studied previously in [20-27]. The other is a nonlinear system already exploited in fuzzy modeling [28-33].

4.1 Gas furnace process

The delayed terms of methane gas flow rate u(t) and carbon dioxide density y(t) such as u(t-3), u(t-2), u(t-1), y(t-3), y(t-2), and y(t-1)are used as input variables to the EA-based SOPNN. The actual system output y(t) is used as target output variable for this model. We choose the input variables of nodes in the 1st layer from these input variables. The total data set consisting of 296 input-output pairs is split into two parts. The first one (consisting of 148 pairs) is used for training. The remaining part of the data set serves

as a testing set. Using the training data set, the coefficients of the polynomial are estimated using the standard LSE. The performance index is defined as the mean squared error

$$PI(EPI) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{5}$$

where y_i is the actual system output, $\hat{y}_i$ is the estimated output of each node, and m is the number of data.

The design parameters of EA-based SOPNN for modeling are shown in Table 5. In the 1st layer, 20 chromosomes are generated and evolved during 40 generations, where each chromosome in the population is defined as corresponding node. So 20 nodes (PDs) are produced in the 1st layer based on the EA operators. All PDs are estimated and evaluated using the training and testing data sets, respectively. They are also evaluated by a fitness function and ranked according to their fitness value. We choose nodes as many as a predetermined number *w* from the highest ranking node, and use their outputs as new input variables to the nodes in the next layer. In other words, The chosen PDs (*w* nodes) must be preserved for the design of the next layer and the outputs of the preserved PDs serve as inputs to the next layer. The value of *w* is different from each layer, which is also shown in Table 5. This procedure is repeated for the 2nd layer and the 3rd layer.

Table 5. Design parameters of EA-based SOPNN for modeling.

Parameters	1st layer	2nd layer	3rd layer
Maximum generations	40	60	80
Population size:(*w)*	20:(15)	60:(50)	80
String length	11	20	55
Crossover rate (P_c)	0.85		
Mutation rate (P_m)	0.05		
Weighting factor: θ	0.1~0.9		
Type (order)	1~3		

w: the number of chosen nodes whose outputs are used as inputs to the next layer

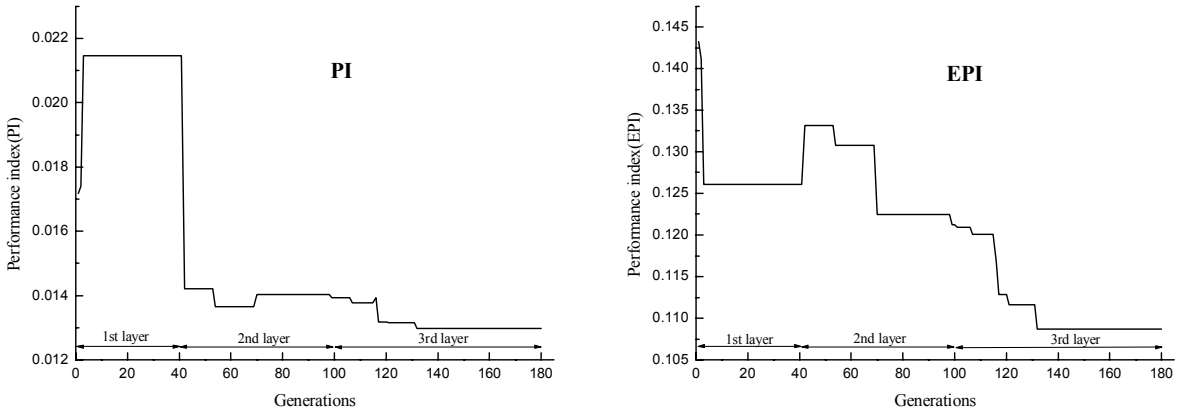
Table 6 summarizes the values of the performance index, PI and EPI, of the proposed EA-based SOPNN according to weighting factor. These values are the lowest value in each layer. The overall lowest value of

the performance index is obtained at the third layer when the weighting factor is 0.5. If this model is designed to have the fourth or higher layer, the performance values come to much lower, but the large computation time is required and the model has much complex network size.

Table 6. Values of performance index of the proposed EA-based SOPNN.

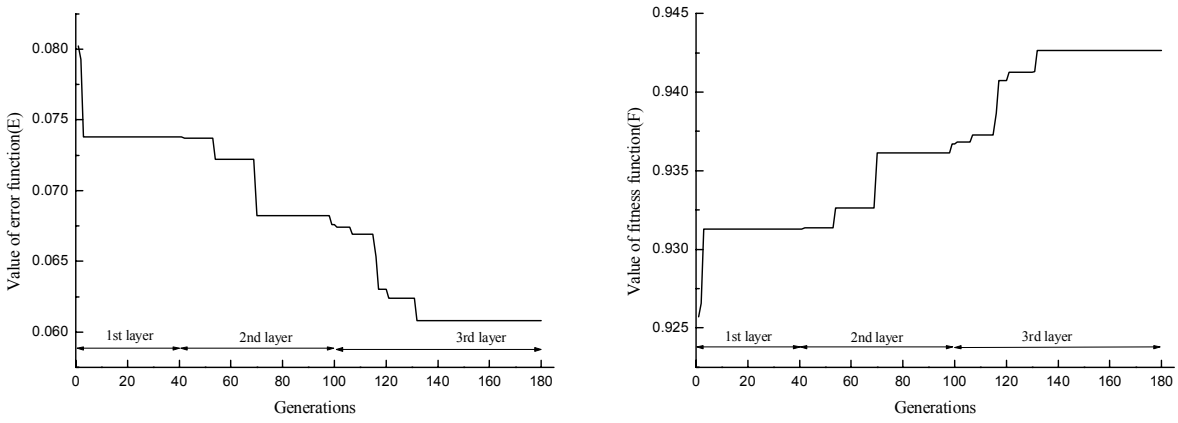
Weighting factor (θ)	1st layer		2nd layer		3rd layer	
	PI	EPI	PI	EPI	PI	EPI
0.1	0.0214	0.1260	0.0200	0.1231	0.0199	0.1228
0.25	0.0214	0.1260	0.0149	0.1228	0.0145	0.1191
0.5	0.0214	0.1260	0.0139	0.1212	0.0129	0.1086
0.75	0.0214	0.1260	0.0139	0.1293	0.0138	0.1235
0.9	0.0173	0.1411	0.0137	0.1315	0.0129	0.1278

Fig. 6 depicts the trend of the performance index values produced in successive generations of the EA when the weighting factor θ is 0.5. Fig. 7 illustrates the values of error function and fitness function in successive EA generations when θ =0.5.



(a) performance index for the training data set (b) performance index for the testing data set

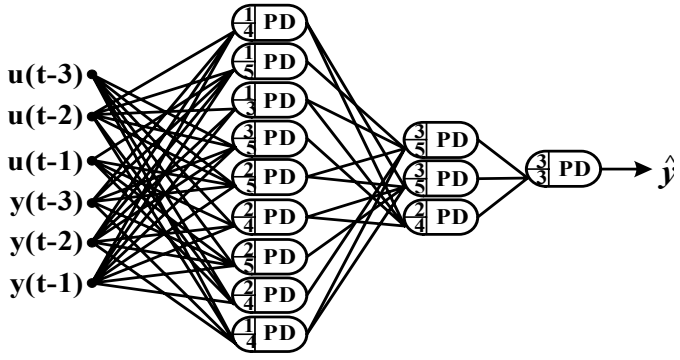
Fig. 6. Trend of performance index values with respect to generations through layers (θ=0.5)



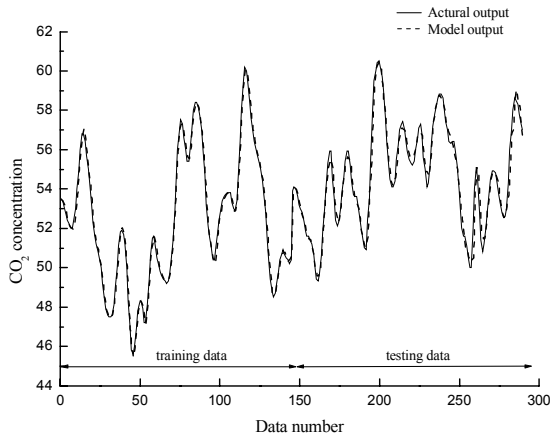
(a) error function (E) (b) fitness function (F)

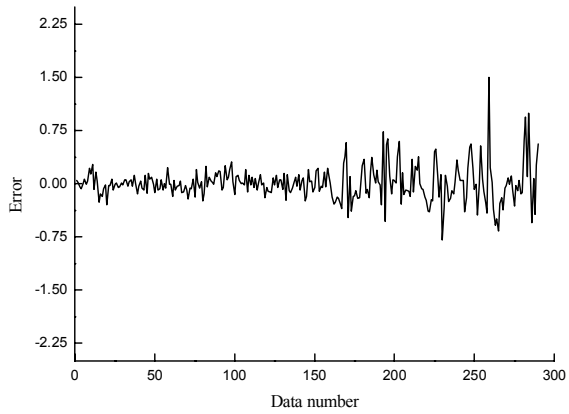
Fig. 7. Values of the error function and fitness function with respect to the successive generations (θ=0.5)

Fig. 8 shows the proposed EA-based SOPNN model with 3 layers and its identification performance when the θ =0.5. The model output follows the actual output very well. Where the values of the performance index of the proposed method are equal to PI=0.012, EPI=0.108, respectively.



(a) Proposed EA-based SOPNN model with 3 layers

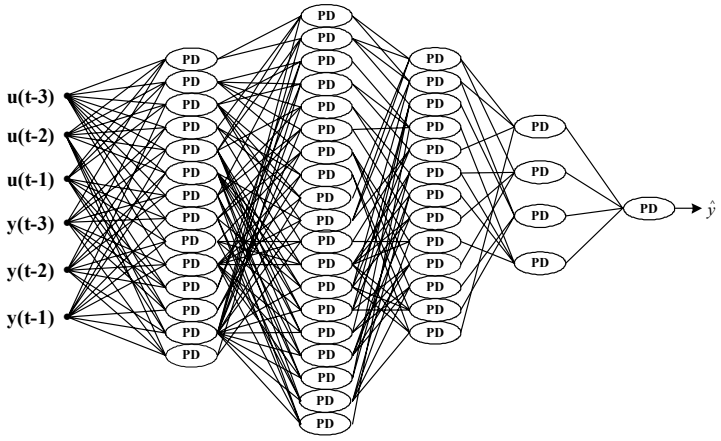




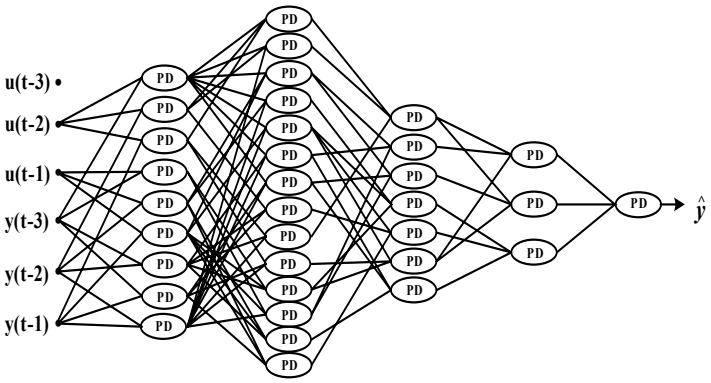
(b) actual output versus model output (c) error

Fig. 8. Proposed EA-based SOPNN model with 3 layers and its identification performance (θ=0.5)

For the comparison of network size of the proposed EA-based SOPNN with that of conventional SOPNN, conventional SOPNN models are visualized in Fig. 9. The structure of the EA-based SOPNN is much simpler than the conventional SOPNN in terms of number of nodes and layers. In addition, the performance of the EA-based SOPNN provides comparable results. Also, EA-based model outperforms the existing identification models. The results of the basic SOPNN & Case 1 in Fig. 9 (a) are obtained in the 5th layer when using 4 inputs and Type 3 to every node in all layers, (that are quantified as PI=0.012, EPI=0.084). The results of the modified SOPNN and Case 2 in Fig. 9 (b) (PI=0.016, EPI=0.101) have been reported when using 2 inputs and Type 1 to every node in the 1st layer and 3 inputs and Type 2 to every node in the 2nd layer or higher.



(a) Basic SOPNN & Case 1



(b) Modified SOPNN & Case 2

Fig. 9. Conventional SOPNN models with 5 layers

Table 7. Values of performance index of some identification models.

Model		Performance index		
		PI	PI	EPI
Tong's model[20]		0.469		
Sugeno and Yasukawa's model[21]		0.190		
Xu's model[22]		0.328		
Pedrycz's model[23]		0.320		
Leski and Czogala's model[24]		0.047		
Kang's model[25]		0.161		
Kim's model[26]			0.034	0.244
Lin and Cunningham's model[27]			0.071	0.261
Kim's model [12]			0.013	0.126
SOPNN (5 layers) [13]	Basic &Case 1		0.012	0.084
	Modified&Case2		0.016	0.101
EA-based SOPNN (3 layers)			0.012	0.108

Table 7 provides a comparison of the proposed model with other techniques being already proposed in

the literature. The comparison is realized on the basis of the same performance index for the training and testing data set. Additionally, PI denotes a performance index of the models for the entire data set (not being split into a training and testing set). PI denotes a performance index of the model for the training data set while EPI for the testing data. It is obvious that the proposed architecture outperforms other models both in terms of their accuracy and higher generalization capabilities.

5.2. A Three-Input Nonlinear Function

In this example, we will demonstrate how the proposed EA-based SOPNN model can be employed to identify the highly nonlinear function. The performance of this model will be compared with earlier works. The function to be identified is a three-input nonlinear function given by (6)

$$y = (1 + x_1^{0.5} + x_2^{-1} + x_3^{-1.5})^2 \tag{6}$$

which is widely used by Takagi and Hayashi[28], Sugeno and Kang[29], and Kondo[30] to test their modeling approaches. Table 8 shows 40 pairs of the input-output data obtained from (6) [32]. The input x_4 is a dummy variable which has no relation to (6). The data on Table 8 is divided into training data set (Nos. 1-20) and testing data set (Nos. 21-40). To compare the performance, the same performance index, average percentage error (APE) adopted in [28-32] is used.

$$APE = \frac{1}{m}\sum_{i=1}^{m}\frac{|y_i - \hat{y}_i|}{y_i} \times 100 \qquad (\%) \tag{7}$$

where m is the number of data pairs and y_i and $\hat{y}_i$ are the i-th actual output and model output, respectively.

Again, a series of comprehensive experiments was conducted and the results are summarized in the same way as before. The design parameters of EA-based SOPNN in each layer are shown in Table 9.

Table 8. Input-output data of three-input nonlinear function.

Training data (1-20)						Testing data (21-40)					
No.	x_1	x_2	x_3	x_4	y	No.	x_1	x_2	x_3	x_4	y
1	1	3	1	1	11.11	21	1	1	5	1	9.545
2	1	5	2	1	6.521	22	1	3	4	1	6.043
3	1	1	3	5	10.19	23	1	5	3	5	5.724
4	1	3	4	5	6.043	24	1	1	2	5	11.25
5	1	5	5	1	5.242	25	1	3	1	1	11.11
6	5	1	4	1	19.02	26	5	5	2	1	14.36
7	5	3	3	5	14.15	27	5	1	3	5	19.61
8	5	5	2	5	14.36	28	5	3	4	5	13.65
9	5	1	1	1	27.42	29	5	5	5	1	12.43
10	5	3	2	1	15.39	30	5	1	4	1	19.02
11	1	5	3	5	5.724	31	1	3	3	5	6.38
12	1	1	4	5	9.766	32	1	5	2	5	6.521
13	1	3	5	1	5.87	33	1	1	1	1	16
14	1	5	4	1	5.406	34	1	3	2	1	7.219
15	1	1	3	5	10.19	35	1	5	3	5	5.724
16	5	3	2	5	15.39	36	5	1	4	5	19.02
17	5	5	1	1	19.68	37	5	3	5	1	13.39
18	5	1	2	1	21.06	38	5	5	4	1	12.68
19	5	3	3	5	14.15	39	5	1	3	5	19.61
20	5	5	4	5	12.68	40	5	3	2	5	15.39

Table 9. Design parameters of EA-based SOPNN for modeling .

Parameters	1st layer	2nd layer	3rd layer
Maximum generations	40	60	80
Population size:(*w)*	20:(15)	60:(50)	80
String length	8	20	55
Crossover rate (P_c)	0.85		
Mutation rate (P_m)	0.05		
Weighting factor: θ	0.1~0.9		
Type (order)	1~3		

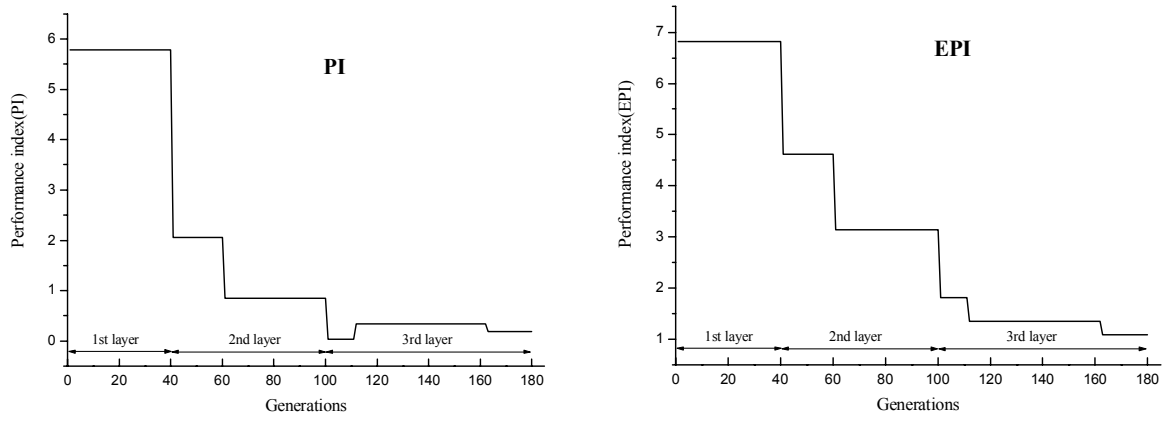
w: the number of chosen nodes whose outputs are used as inputs to the next layer

The simulation results of the EA-based SOPNN are summarized in Table 10. The overall lowest values of the performance index, PI=0.188 EPI=1.087, are obtained at the third layer when the weighting factor (θ) is 0.25.

Table 10. Values of performance index of the proposed EA-based SOPNN model.

Weighting factor	1st layer		2nd layer		3rd layer	
	PI	EPI	PI	EPI	PI	EPI
0.1	5.7845	6.8199	2.3895	3.3400	2.2837	3.1418
0.25	5.7845	6.8199	0.8535	3.1356	0.1881	1.0879
0.5	5.7845	6.8199	1.6324	5.5291	1.2268	3.5526
0.75	5.7845	6.8199	1.9092	4.0896	0.5634	2.2097
0.9	5.7845	6.8199	2.5083	5.1444	0.0002	4.8804

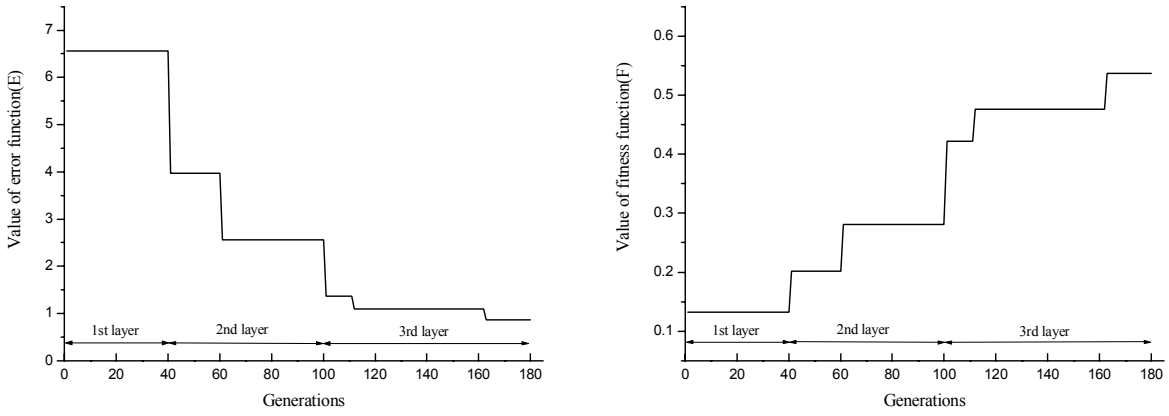
Fig. 10 illustrates the trend of the performance index values produced in successive generations of the EA when the weighting factor θ is 0.25.



(a) performance index for the training data set (b) performance index for the testing data set

Fig. 10. Trend of performance index values with respect to generations through layers (θ =0.25)

Fig. 11 shows the values of error function and fitness function in successive EA generations when the θ is 0.25.



(a) error function (E) (b) fitness function (F)

Fig. 11. Values of the error function and fitness function with respect to the successive generations (θ =0.25)

Fig. 12 depicts the proposed EA-based SOPNN model with 3 layers when the θ is 0.25. The structure of EA-based SOPNN is very simple and has a good performance. But for the conventional SOPNN, it is difficult to structure the model for this nonlinear function. That is why a few number of input candidates are considered [33].

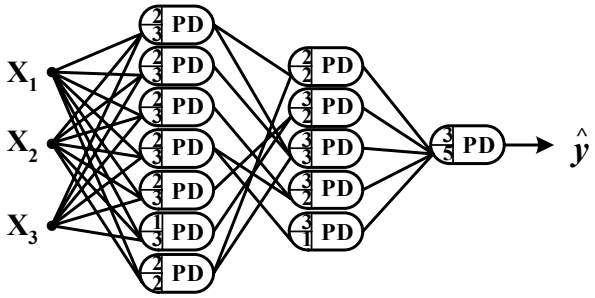
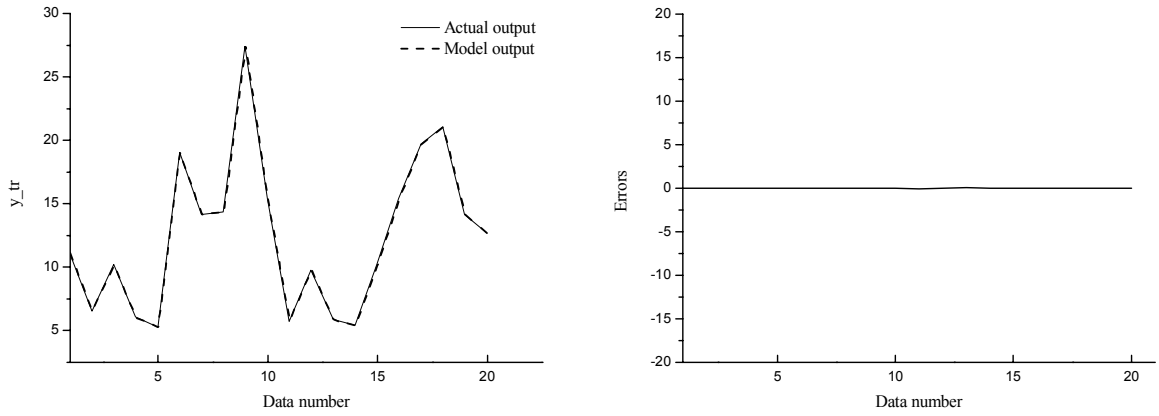
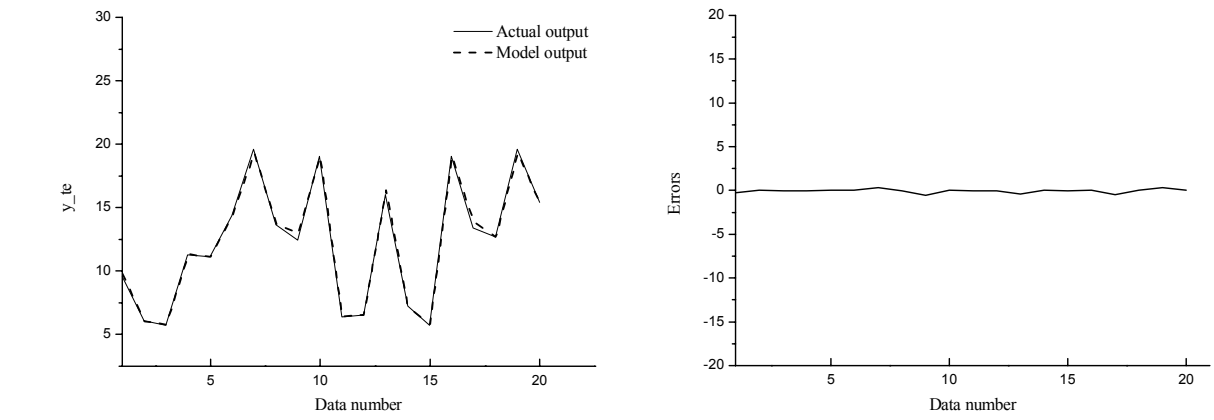


Fig. 12. Structure of the EA-based SOPNN model with 3 layers (θ =0.25)

Fig. 13 shows the identification performance of the proposed EA-based SOPNN and its errors when the θ is 0.25. The output of the EA-based SOPNN follows the actual output very well.



(a) actual output versus model output of training data set (b) errors of (a)



(c) actual output versus model output of testing data set (d) errors of (c)

Fig. 13. Identification performance of EA-based SOPNN model with 3 layers and its errors

Table 11 shows the performance of the proposed EA-based SOPNN model and other models studied in the literature. The experimental results clearly reveal that the proposed model outperforms the existing models both in terms of better approximation capabilities (PI) as well as superb generalization abilities (EPI). But the conventional SOPNN cannot be applied to the identification of this example.

Table 11. Performance comparison of various identification models.

Model		APE	
		PI (%)	EPI (%)
GMDH model[30]		4.7	5.7
Fuzzy model [29]	Model 1	1.5	2.1
	Model 2	0.59	3.4
FNN [32]	Type 1	0.84	1.22
	Type 2	0.73	1.28
	Type 3	0.63	1.25
GD-FNN [31]		2.11	1.54
Conventional SOPNN [13]		***Impossible***	
EA-based SOPNN		0.188	1.087

5. Conclusions

In this paper, we propose a new design methodology of SOPNN using evolutionary algorithm, which is called as the EA-based SOPNN and study properties of EA-based SOPNN. We can see that the EA-based SOPNN is a sophisticated and versatile architecture which can construct models for limited data set and poorly defined complex problems. Moreover, the architecture of the model is not predetermined, but can be self-organized automatically during the design process. The conflict between overfitting and generalization can be avoided by using fitness function with weighting factor. The experimental results show that the proposed EA-based SOPNN is superior to the conventional SOPNN models as well as other previous models in terms of the modeling performance.

References

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *IEEE Trans. Neural Netw.*, Vol. 2, pp. 359-366, Mar. 1989.

[2] T. Chen and H. Chen, "Approximation capability to functions of several variables, nonlinear functions, and operators by radial basis function neural networks," *IEEE Trans. Neural Netw.*, Vol. 6, pp. 904-910, July. 1995.

[3] K. Li, "Approximation theory and recurrent networks," in *Proc. IJCNN*, Vol. Ⅱ, pp. 266-271, 1992.

[4] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.,* Vol. 22, pp. 1414-1427, June, 1992.

[5] ____, "Fuzzy basis function, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Netw.*, Vol. 3, pp. 807-814, Sep., 1992.

[6] A. G. Ivakhnenko, "Polynomial theory of complex systems", *IEEE Trans. Syst., Man, Cybern.,* Vol. SMC-1, No. 1, pp. 364-378, 1971.

[7] A. G. Ivakhnenko and N. A. Ivakhnenko, "Long-term prediction by GMDH algorithms using the unbiased criterion and the balance-of-variables criterion," *Sov. Automat. Contr.*, Vol. 7, pp. 40-45, 1974.

[8] ____, "Long-term prediction by GMDH algorithms using the unbiased criterion and the balance-of-variables criterion, part 2," *Sov. Automat. Contr.*, Vol. 8, pp. 24-38, 1975.

[9] A. G. Ivakhnenko, V. N. Vysotskiy, and N. A. Ivakhnenko, "Principal version of the minimum bias criterion for a model and an investigation of their noise immunity," *Sov. Automat. Contr.*, Vol. 11, pp. 27-45, 1978.

[10] A. G. Ivakhnenko, G. I. Krotov, and N. A. Ivakhnenko, "Identification of the mathematical model of a

complex system by the self-organization method," in *Theoretical Systems Ecology: Advances and Case Studies*, E. Halfon, Ed. New York: Academic, 1970, ch. 13

[11] S. J. Farlow, *Self-Organizing Methods in Modeling, GMDH Type-Algorithms*, New York: Marcel Dekker, 1984.

[12] D. W. Kim, " *Evolutionary Design of Self-Organizing Polynomial Neural Networks*," Master's thesis, Dept. Control Instrum., Wonkwang Univ., 2002(in Korean).

[13] S. K. Oh and W. Pedrycz, "The design of self-organizing Polynomial Neural Networks," *Inf. Sci.*, Vol. 141, pp. 237-258, 2002.

[14] Y. Shi, R. Eberhart, and Y. Chen, "Implementation of Evolutionary Fuzzy Systems," *IEEE Trans. Syst., Man, Cybern.,* Vol. 7, No. 2, pp. 109-119, April, 1999.

[15] K. Kristinnson and G.A. Dumont, "System identification and control using genetic algorithms," *IEEE Trans. Syst., Man, Cybern.,* Vol. 22, No. 5, pp. 1033-1046, 1992.

[16] S. Uckun, S. Bagchi, and K. Kawamura, "Managing genetic search in job shop scheduling," *IEEE Expert*, Vol. 8, No. 5, pp. 15-24, 1993.

[17] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning. Reading*, MA: Addison-Wesley, 1989.

[18] S.Y. Kung and J.S. Taur, "Decision-based neural networks with signal/image classification applications," *IEEE Trans. Neural Netw.*, Vol. 6, pp. 170-181, Jan. 1995.

[19] G.E.P. Box and F.M. Jenkins, *Time Series Analysis* : *Forecasting and Control* 2nd ed. Holden-day, 1976.

[20] R.M. Tong, "The evaluation of fuzzy models derived from experimental data", *Fuzzy Sets Syst.*, Vol.13, pp.1-12, 1980.

[21] M. Sugeno and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling", *IEEE Trans. Fuzzy Syst.*, Vol. 1, No. 1, pp. 7-31, 1993.

[22] C.W. Xu, and Y. Zailu, "Fuzzy model identification self-learning for dynamic system, *IEEE Trans. Syst., Man, Cybern.,* Vol. SMC-17, No.4, pp.683-689, 1987.

[23] W. Pedrycz, "An identification algorithm in fuzzy relational system", *Fuzzy Sets Syst.*, Vol. 13, pp.153-167, 1984.

[24] J. Leski, and E. Czogala, "A new artificial neural networks based fuzzy inference system with moving consequents in if-then rules and selected applications", *Fuzzy Sets Syst.*, Vol. 108, 289-297, 1999.

[25] S.J. Kang, C.H. Woo, H.S. Hwang, and K.B. Woo, "Evolutionary Design of Fuzzy Rule Base for Nonlinear System Modeling and Control," *IEEE Trans. Fuzzy Syst.*, Vol. 8, No. 1, Feb., 2000.

[26] E. Kim, H. Lee, M. Park, M. Park, "A simple identified Sugeno-type fuzzy model via double clustering," *Inf. Sci.*, Vol. 110, pp. 25-39, 1998.

[27] Y. Lin, G.A. Cunningham Ⅲ, "A new approach to fuzzy-neural modeling", *IEEE Trans. Fuzzy Syst.*, Vol. 3, No. 2, pp. 190-197, 1995.

[28] H. Takagi and I. Hayashi, "NN-driven fuzzy reasoning," *Int. J. Approx. Reasoning*, Vol. 5, No. 3, pp. 191-212, 1991.

[29] M. Sugeno and G. T. Kang, "Structure identification of fuzzy model," *Fuzzy Sets Syst.*, Vol. 28, pp. 15-33, 1988.

[30] T. Kondo, "Revised GMDH algorithm estimating degree of the complete polynomial," *Tran. Soc. Instrum. Control Eng.*, Vol. 22, No. 9, pp. 928-934, 1986(in Japanese)

[31] S. Wu, M.J. Er, and Y. Gao, "A Fast Approach for Automatic Generation of Fuzzy Rules by Generalized Dynamic Fuzzy Neural Networks," *IEEE Trans. Fuzzy Syst.*, Vol. 9, No. 4, pp. 578-594, 2001.

[32] S.I. Horikawa, T. Furuhashi, and Y. Uchikawa, "On Fuzzy modeling Using Fuzzy Neural Networks with the Back-Propagation Algorithm," *IEEE Trans. Neural Netw.*, Vol. 3, No. 5, pp. 801-806, 1992.

[33] Dong-Won Kim, Sung-Kwun Oh, and Hyun-Ki Kim, "A Study on the Self-organizing Fuzzy Polynomial Neural Networks," *Journal of KIEE*, Vol. 11, No. 2, pp. 79-89, 2001.