Fuzzy Sets and Systems 108 (1999) 179–191

www.elsevier.com/locate/fss

Fuzzy system modeling using linear distance rules

Sheng-De Wang*, Chien-Hui Lee

Department of Electrical Engineering, EE Building, Rm. 441, National Taiwan University, Taipei 106, Taiwan

Received July 1996; received in revised form August 1996

Abstract

An approach of determining significant variables for multi-input systems is proposed by using a modified fuzzy rule form, called linear distance rules (LDR). By defining an index, we can know the influences of input signals on output and can determine significant variables from the index. We also propose a simple three-layered generalized neural network to realize the LDRs. Furthermore, this approach can be extended to decompose a multi-input–multi-output (MIMO) system into several simpler multi-input–single-output (MISO) systems. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Fuzzy model; Neural networks

1. Introduction

Takagi and Sugeno [26] proposed a type of fuzzy rules with consequences consisting of linear combination of input signals. In [8,25], they used the Kalman filtering algorithm to update the parameters in both premises and consequences based on the Takagi–Sugeno fuzzy rules. In [8], it is assumed that all the input signals are equally crucial without determining which variables are significant. This could make the updated parameters more than it does require. Appropriately selecting significant variables can help us not only achieve the task of identification easier and even obtain more precise result, since dummy variables may disturb training effect. In [25], Sun sketched fuzzy curves for a MISO system to decide significant variables through human inspection. Using this approach, $n \cdot m$ fuzzy curves must be drawn for an n-input and m-output system, and then select significant variables by investigating the fuzzy curves. This seems a time-consuming task.

For modeling an unknown system, a modified Takagi–Sugeno fuzzy rule form, called *linear distance rules* (*LDR*), is proposed. Based on the LDR, we propose an algorithm to determine significant variables without the need of human inspection, and we use the back-propagation algorithm to complete our identification task. In [8], it is shown that using the Kalman filtering algorithm can be much faster than applying the back-propagation algorithm for modeling a system. However, if there are m consequence parameters to be adapted, an $m \times m$ matrix must be constructed for the Kalman filtering algorithm. In other words, it may require large space to store data if there are a lot of parameters to be updated. This may be even worse when all variables

* Corresponding author.
E-mail address: sdwang@cc.ee.ntu.edu.tw (S.-D. Wang)

are used. This is why we use the back-propagation algorithm instead of the Kalman filtering algorithm. Furthermore, we shall discuss the tasks related to discarding unimportant variables and incorporating a set of fuzzy rules with the same premises into one rule to reduce the size of neural networks. And we can also use the same approach to decompose a complicated MIMO system into several simpler and almost decoupled MISO systems if the solution exists.

2. Motivations

One type of fuzzy rules with singleton consequences for a MISO system is described as

$$\text{Rule i: if } x_1 \text{ is } A_1^i \text{ and } \cdots \text{ and } x_n \text{ is } A_n^i \text{ then } u \text{ is } a_i, \tag{1}$$

where $a_i \in R$. If all the membership functions in the premise are defined as Gaussian functions and their parameters are fixed, by the product operation, the firing strength is in the form of *fuzzy basis function* (*FBF*) and can be written as

$$g_i\left(\frac{\|\boldsymbol{x}-c_i\|}{\sigma_i}\right) = \prod_{j=1}^{n} \mu_{A_j^i}(x_j), \quad i=1,2,\ldots,N, \tag{2}$$

where $\boldsymbol{x}$ is the input vector, $\mu_{A_j^i}(x_j)$ are the Gaussian membership functions, $c_i=(c_1^i,c_2^i,\ldots,c_n^i)\in R^n$ are the centers of the ith rule and N is the number of fuzzy rules. Then the output of the fuzzy system can be written as an FBF expansion:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i g_i\left(\frac{\|\boldsymbol{x}-c_i\|}{\sigma_i}\right), \tag{3}$$

where w_i is a real number and equivalent to the singleton consequence of a fuzzy rule.

In fact, fuzzy inference can be viewed as an interpolation of completely or partially matching fuzzy rules. One property of interpolation is that the result must be something in between the largest and smallest values. Similarly, the fuzzy reasoning result must be smaller than the maximal consequence and larger than the minimal consequence of firing rules. Let us consider a function $y=f(x)$ to be identified, and we have a set of I/O data pair. The function $f(x)$, x is in some closed bounded set, could contain several local minima and maxima. If the centers of predefined fuzzy rules are located on the maxima or minima, the interpolation result will be better; otherwise, the value of the approximation result cannot reach the extremities. Unfortunately, in general, we cannot precisely pinpoint where the local minima and maxima of an unknown function $f(x)$ are located just from the I/O data. To find the local minima and maxima of a function is another research subject. So it is impossible for users to set the centers of fuzzy rules exactly on these peaks or valleys. If we regularly define the fuzzy linguistic terms by equally dividing the range of each variable, the inference result, from these fuzzy rules, would have large error in the maximal or minimal values. So some learning algorithms try to train both parameters in the premise and consequence to change the centers of fuzzy rules [14] as well as the weightings of firing strengths.

Consider a simple SISO function $f(x)=x+2\sin(x)$, where $x\in[0,2\pi]$. The function has one local maximum ($x=\pi/2$) and one local minimum ($x=3\pi/2$). If we give three SISO fuzzy rules as (1), their centers of the premise are defined as $0,\pi,2\pi$, respectively, and their consequences are given in advance and kept fixed, i.e.

$$\begin{aligned}
&\text{if } x \text{ is near zero,} && \text{then } y=0+2\sin(0)=0,\\
&\text{if } x \text{ is near } \pi, && \text{then } y=\pi+2\sin(\pi)=\pi,\\
&\text{if } x \text{ is near } 2\pi, && \text{then } y=2\pi+2\sin(2\pi)=2\pi.
\end{aligned}$$

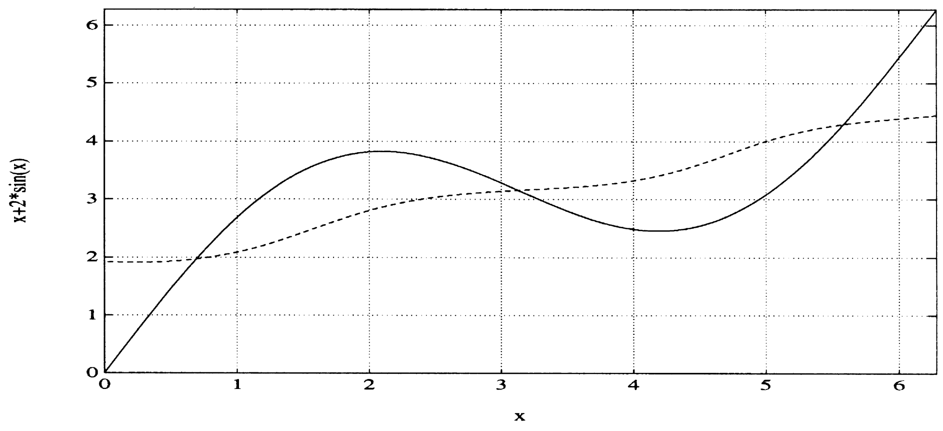

Fig. 1. Modeling of a $x + 2\sin(x)$ function with three rules with singleton consequences. The desired output is shown by the solid line and the actual output by the dashed line.

It is obvious that the inference result of using the three fuzzy rules cannot well mimic the function $f(x)$ for $x \in [0, 2\pi]$. Now, we assume the consequences are the only adjustable parameters, the result obtained by using the back-propagation training algorithm is shown in Fig. 1. The approximation capability through training of three fuzzy rules with singleton consequences still cannot emulate the function. Since the consequences of fuzzy rules are fuzzy singletons, the peak or valley value between two centers of rules cannot be reached through the fuzzy reasoning or interpolation. To solve such problem, there are several methods: (1) add more fuzzy rules; (2) not only adjust the consequences but also update the parameters in the antecedence of fuzzy rules through a learning algorithm, such as back propagation, in order to move the centers of rules to the suitable positions; however, this must propagate the error messages back to the prior layer and will slow down the training process; (3) concern the consequences of fuzzy rules with input signals. In this paper, we shall focus on the third point.

3. An introduction to Takagi–Sugeno fuzzy rules

In the last section, we demonstrate some disadvantages of fuzzy rules with singleton consequences. For overcoming the disadvantages, Takagi–Sugeno fuzzy rules provide an alternative approach. Such type of fuzzy rules can be expressed as

$$\begin{aligned} &\text{Rule i:} \quad \text{if} \quad (x_1 \text{ is } A_1^i \text{ and } \cdots \text{ and } x_n \text{ is } A_n^i),\\ &\qquad\qquad \text{then} \quad (u \text{ is } a_0^i + a_1^i x_1 + \cdots + a_n^i x_n), \end{aligned} \tag{4}$$

where the coefficients $a_0^i, a_1^i, \ldots, a_n^i$ are adjustable. The inference procedure in the premise part is the same as the fuzzy basis function expansion. The difference between fuzzy rules with singleton consequences and Takagi–Sugeno fuzzy rules is only in the action part. The former is just a real number, while the latter is a linear combination of input signals. From (4) if we set $a_j^i = 0$, $\forall j = 1, 2, \ldots, n$, the fuzzy rules will be the same as (1). In fact, the span of fuzzy rules as (1) is, indeed, a subset of Takagi–Sugeno fuzzy rules.

Fig. 2 shows that the inference result of two Takagi–Sugeno fuzzy rules can be out of range $[c_1, c_2]$, where c_1 and c_2 are the smaller and larger consequences of given two fuzzy rules, respectively. Let us consider identifying the previous function $f(x) = x + 2\sin(x)$ by using Takagi–Sugeno fuzzy rules. Fig. 3 shows that

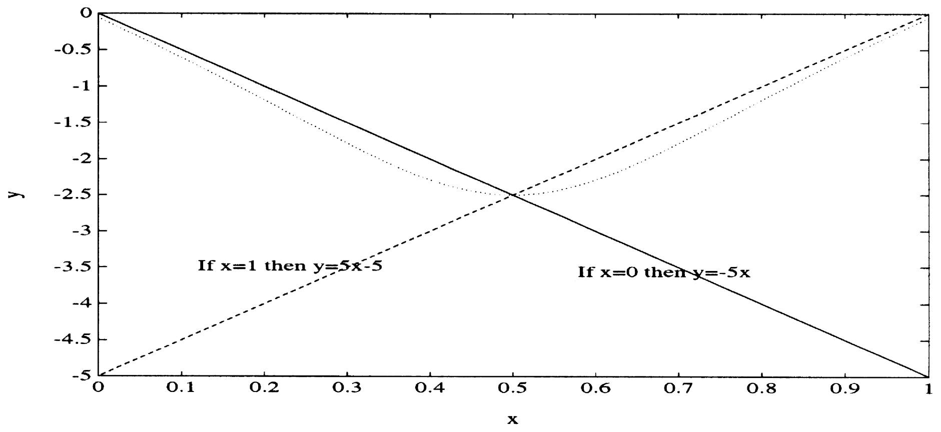

Fig. 2. Inference result of two Takagi–Sugeno fuzzy rules.

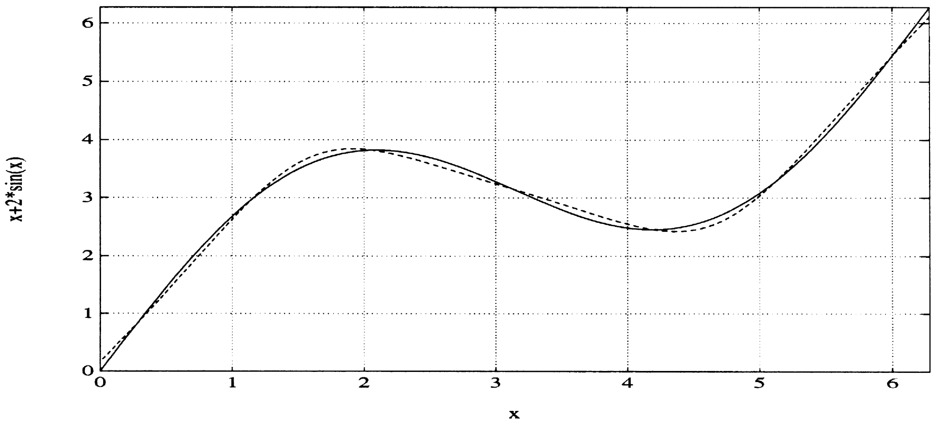

Fig. 3. Modeling of a $x + 2\sin(x)$ function with three Takagi–Sugeno fuzzy rules. The desired output is shown by the solid line and the actual output by the dashed line.

it can emulate the function by using only three Takagi–Sugeno fuzzy rules whose centers are also located on $0, \pi$ and 2π, respectively.

Actually, it is possible for Takagi–Sugeno fuzzy rules to produce one maximum and one minimum between two consecutive fuzzy rules by appropriately designing consequence parameters. So it is expected that using less fuzzy rules of form (4) to mimic a function than applying fuzzy rules of form (1).

From Figs. 1–3, we have demonstrated a simple example and compare with fuzzy rules with singleton consequences to explain why we use Takagi–Sugeno fuzzy rules. The disadvantages of (1) we have mentioned can be overcome if we apply Takagi–Sugeno fuzzy rules instead. In this paper, we use a class of membership functions called Gaussian-like function (GLF). The GLF, $g_l(x)$, is defined as

$$g_l(x) = \exp\left(-\left(\frac{x-c}{2\sigma}\right)^2\right), \tag{5}$$

where c is the center of the fuzzy set.

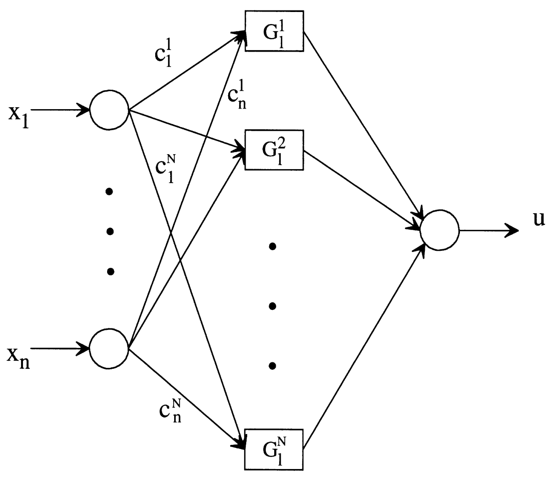

Fig. 4. The architecture of generalized neural networks (GNN).

An introduction to the architecture of generalized neural networks (GNN) was presented in [8,25]. Here, we modify the structure and use only three layers in our proposed GNN as shown in Fig. 4. The circle nodes only perform some known special function and have no parameters to be tuned. However, the square nodes represent there are some adjustable parameters in it. In the following, the processing functions of the proposed GNN are explained. The first node distributes the input signal to the network just like traditional neural networks. The weights in the second layer depict the centers of premises of fuzzy rules. The inputs to the second layer are the distances between the first-layer outputs and the weights in the layer 2. If we take the product operation in computing the firing strength, we have the following matching point of the ith fuzzy rule:

$$\mu_i(\boldsymbol{x}) = g_l(x_1 - c_1^i) \cdot g_l(x_2 - c_2^i) \cdots g_l(x_n - c_n^i) \tag{6}$$

and the defuzzification output of fuzzy inference is the linear combination of firing strength [22] as follows:

$$o = \sum_i o_i = \sum_{i=1}^{N} \mu_i(a_0^i + a_1^i x_1 + \cdots + a_n^i x_n), \tag{7}$$

where o is the output of fuzzy reasoning, N is the number of fuzzy rules, o_i and μ_i are the contribution to the output and the firing strength of the ith fuzzy rule, respectively.

Nevertheless, we must compute the linear combination of input signals by using Takagi–Sugeno fuzzy rules. In [8], the computing processes are not assigned to the GNN and they are considered as extraneous signals to nodes in the fourth layer.

For simplifying the GNN, we slightly modify the rule expression, called *linear distance rule* (*LDR*), and the ith rule is shown as

$$\begin{aligned} \text{Rule } i: \quad & \text{if} \quad (x_1 \text{ is } A_1^i \text{ and } \cdots \text{ and } x_n \text{ is } A_n^i), \\ & \text{then} \quad (u \text{ is } a_0^i + a_1^i(x_1 - c_1^i) + \cdots + a_n^i(x_n - c_n^i)), \end{aligned} \tag{8}$$

where c_k^i is the center of the membership function A_k^i. Then the third-layer nodes complete the defuzzification task, i.e.

$$\begin{aligned} o &= \sum_{i=1}^{N} g_l(x_1 - c_1^i) \cdots g_l(x_n - c_n^i) \cdot (a_0^i + a_1^i(x_1 - c_1^i) + \cdots + a_n^i(x_n - c_n^i)) \\ &= \sum_{i=1}^{N} \mu_i \cdot (a_0^i + a_1^i(x_1 - c_1^i) + \cdots + a_n^i(x_n - c_n^i)) \\ &= \sum_{i=1}^{N} G_l^i(x_1 - c_1^i, x_2 - c_2^i, \ldots, x_n - c_n^i), \end{aligned} \tag{9}$$

where G_l^i is the function that the ith node of layer 2 achieves and $a_0^i, a_1^i, \ldots, a_n^i$ are the adjustable parameters in layer 2. From (9), we can find the functions that the second-layer nodes perform are functions of distances between input signals and centers of fuzzy rules. Finally, the node in the third layer just completes a summation function that corresponds to the defuzzification process. The advantages of the type of rules (8) over that of (4) is that we can acquire the fuzzy rules with singleton consequences easily without computation. This can help us incorporate the system with fuzzy rules from experts, and we can know the influence of displacement $(x - c^i)$ on the output.

For system identification, we apply the supervised back-propagation learning method. First, define an error function to be the square sum of difference between desired and actual outputs, i.e.

$$E = \frac{1}{2} \sum (o^{\mathrm{d}} - o)^2, \tag{10}$$

where o^{d} and o are the desired output and actual output, respectively. Our objective is to minimize the error function. The general learning rule using the concept of gradient descent is

$$\Delta a_j^i \propto -\frac{\partial E}{\partial a_j^i} \tag{11}$$

and

$$a_j^i(t+1) = a_j^i(t) + \eta_j \left(-\frac{\partial E}{\partial a_j^i} \right), \tag{12}$$

where η_j is the learning rate.

For simplicity, we keep the parameters in the second layer unchanged and only update the consequence parameters. In the condition part, we first decide how many terms for each variable, and equally divide the range of each variable.

In the following, we shall simply derive the updating rule for the coefficients in the consequence of fuzzy rules. Assume a_j^i are the adjustable parameters, the updating rules are

$$\frac{\partial E}{\partial a_j^i} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial o_i} \cdot \frac{\partial o_i}{\partial a_j^i} = \begin{cases} (o^{\mathrm{d}} - o) \cdot \dfrac{\mu_i}{\sum \mu_i} & \text{if } j = 0, \\ (o^{\mathrm{d}} - o) \cdot \dfrac{\mu_i}{\sum \mu_i} x_j & \text{otherwise.} \end{cases} \tag{13}$$

From Eq. (13), we can obtain the updating rule

$$a_j^i(t+1) = a_j^i(t) + \eta_j\left(-\frac{\partial E}{\partial a_j^i}\right) = \begin{cases} a_j^i(t) + \eta_0(o^{\mathrm{d}} - o)\cdot\dfrac{\mu_i}{\sum \mu_i} & \text{if } j = 0, \\ a_j^i(t) + \eta_j(o^{\mathrm{d}} - o)\cdot\dfrac{\mu_i}{\sum \mu_i}x_j & \text{otherwise.} \end{cases} \tag{14}$$

If the modeling accuracy cannot satisfy us, we can back propagate error messages to the previous layer to update the premises. The derivation processes of updating rule for the previous layer can be found in papers discussing back-propagation [14,15]. We shall not rewrite these mathematical equations here. By this way, we must consider some influences of moving centers of fuzzy rules, e.g., the property of completeness.

4. Selecting significant variables

For an unknown system, it could contain many associated input and state variables. If all the variables are considered, the number of fuzzy rules will increase exponentially. For example, a system with five input variables and each variable is assigned three terms, i.e. the number of total rules is 125. If we choose Takagi–Sugeno fuzzy rules or LDRs, the adjustable parameters will also increase by several orders.

For fuzzy-rule-based identification task, we must take two main issues into account. One is the modeling accuracy that should satisfy our expectation; the other is the number of fuzzy rules should be small. By the Stone–Weierstrass theorem, the fuzzy basis function expansions can be a universal approximator [28] and the modeling error can be arbitrarily small if there are a considerable number of fuzzy rules; however, in real applications the number of fuzzy rules is finite. Therefore, it is generally a dilemma between the two above-mentioned subjects – the accuracy and the size of rule base. Based on this concept, developing an approach to select significant variables becomes an important issue to solving the identification problems by using fuzzy systems.

4.1. Algorithm

Firstly, we choose LDRs as our fuzzy rules and the updating rules of the back-propagation (BP) algorithm should be modified as

$$\frac{\partial o}{\partial a_j^i} = \begin{cases} (o^{\mathrm{d}} - o)\cdot\dfrac{\mu_i}{\sum \mu_i} & \text{if } j = 0, \\ (o^{\mathrm{d}} - o)\cdot\dfrac{\mu_i}{\sum \mu_i}(x_j - c_j^i) & \text{otherwise.} \end{cases} \tag{15}$$

From the experience of applying BP, the value of the error function decreases rapidly in the beginning. After that, the error will be decreased very very slowly. Therefore, we propose a two-phase learning procedure. In phase 1, our main objective is to select significant variables. If the number of training iterations is less than some integer I_{t} set by users, we only adjust the consequences of fuzzy rules and keep the premise part unchanged. While the iteration number reaches I_{t}, we shall make a decision to choose the significant signals and shrink the neural network such that its inputs are just the selected significant variables.

To shrink the architecture of neural networks, we alloy the trained fuzzy rules by average, e.g., originally we have four fuzzy rules as below

	x_2 is near 0	x_2 is near 1
x_1 is near 0	$a_0^1 + a_1^1(x_1 - c_1^1) + a_2^1(x_2 - c_1^2)$	$a_0^2 + a_1^2(x_1 - c_1^1) + a_2^2(x_2 - c_2^2)$
x_1 is near 1	$a_0^3 + a_1^3(x_1 - c_2^1) + a_2^3(x_2 - c_1^2)$	$a_0^4 + a_1^4(x_1 - c_2^1) + a_2^4(x_2 - c_2^2)$

Now, we assume that x_1 is the significant variable, the consequences of fuzzy rules can be averaged by the associated rules as

if	then
x_1 is near 0	$\frac{1}{2}(a_0^1 + a_0^2) + \frac{1}{2}(a_1^1 + a_1^2)(x_1 - c_1^1)$
x_1 is near 1	$\frac{1}{2}(a_0^3 + a_0^4) + \frac{1}{2}(a_1^3 + a_1^4)(x_1 - c_2^1)$

After reducing the networks, we can restart the training task. In the second phase, we can transfer error message back not only to the third layer but also to the second layer to update the consequences and centers and widths in the premise part in order to minimize the error function if error cannot be lowered down to a desired value. The training process stops when the error function is less than an acceptable value or the running time reaches a value we defined, which is denoted I_{stop}. In the following, we give a simple example to demonstrate some properties of LDR.

Example 1. Given two rules with only one condition and one action, we wish to complete some mapping $f: x \to y$, where x and y are both scalars:

Rule 1: if x is A_1 then a_1 is $a_0^1 + a_1^1(x - c^1)$

and

Rule 2: if x is A_2 then a_2 is $a_0^2 + a_1^2(x - c^2)$, (16)

where c_i, $i = 1, 2$ are the centers of membership functions A_1 and A_2. In (16), each consequence is a straight line and a_1^1, a_1^2 are the slopes of the lines in the x–y plane. For Rule 1 (or Rule 2), the action will be a_0^1 (or a_0^2), if x is equal to c^1 (or c^2). While a_1^1 and a_1^2 approach zeros, the line $a_0^1 + a_1^1(x_1 - c^1)$ and $a_0^2 + a_1^2(x_2 - c^2)$ are both almost horizontal. For any almost horizontal line, we can use a line with zero slope to approximate it, since we only consider that variables are in a compact set. This situation implies that the input variable x has no influence on function $f(\cdot)$ and the function $f(\cdot)$ can be approximated by fuzzy rules with singleton consequences. Obviously, we can discard the variables whose coefficients are very near to zeros in the linear combination of consequence of each rule.

Whether the coefficients are small enough to be eliminated, sometimes it depends upon human intuition. But for a self-learning machine, we may define an index function called "near-zero index (NZI)" which is described as

$$\text{NZI}_j = \frac{\sum_i |a_j^i|}{\text{the number of rules}}. \tag{17}$$

The range of NZI depends upon the range of the output data and the property of the identified system itself. It is a relative value rather than an absolute one. Therefore, we think if $\text{NZI}_j/\text{NZI}_0$ is less than some pre-defined small value $\varepsilon_{\mathrm{NZI}}$, the signal is regarded as an unimportant variable. Then we construct a neural network with significant variables only. To evaluate the performance of identification, we use a performance index from [13]. It is defined as in the following:

$$\mathrm{PI} = \frac{\sqrt{\sum_{i=1}^{m}(o_i^{\mathrm{d}} - o_i)^2}}{\sum_{i=1}^{m}|o_i|}. \tag{18}$$

If the performance index cannot satisfy our criterion, we can give a smaller $\varepsilon_{\mathrm{NZI}}$ to consider more input signals. The following is the algorithm of the proposed method:

```
BEGIN
  decide how many terms for each variable,
  design $I_t$,
  define initial centers and widths of membership functions in the second layer,
  if iteration times $< I_t$,
    update consequence parameters,
  else if (iteration times $= I_t$),
  {
    calculate NZIs for each input signal,
    if $\mathrm{NZI}_j < \varepsilon_{\mathrm{NZI}}$ discards the $j$th input signal,
    else the signal is considered as a significant variable,
    reconstruct the neural networks for only the significant variables,
  }
  restart training consequence parameters,
  calculate performance index (PI),
  if PI does not satisfy us, cut down $\varepsilon_{\mathrm{NZI}}$ and back to the step of reconstructing neural networks
END
```

To use this approach, it is very important to normalize all the input signals into some interval, for instance, $[-1, 1]$. We give an example in the following:

Assume the consequence of a fuzzy rule can be described as

$$a = a_0 + a_1(x_1 - c_1) + \cdots + a_n(x_n - c_n), \tag{19}$$

where the range of a is known as $[-a^U, a^U]$, x_1 is between -1000 and $+1000$, and let x_i, $i \geqslant 2$, c_i, $i = 1, \ldots, n$ and a_0 be all zeros. Thus, a_1 should be in the interval $[-a^U/1000, a^U/1000]$. If we change the unit of x_1 such that x_1 is between -1 and 1, a_1 should be enlarged in the range $[-a^U, a^U]$. It illustrates that the consequence parameters $a_1, a_2, \ldots, a_n$ would be influenced severely by different units. In other words, x_i with different units will have different coefficients a_i. If x_1 is enlarged ten times, a_1 will be decreased to $\frac{1}{10}$. This will lead to the variation of NZI values if the input signals are not normalized. So it is necessary to map input signals onto some range.

4.2. Decomposition of a MIMO system

An n-input and m-output plant can be intuitively decomposed into m n-input MISO plants. Actually, some inputs only contribute some outputs and have nothing to do with other outputs. In other words, it is possible to decompose into m plants with the number of inputs less than n by properly selecting significant variables.

Some MIMO systems have very complex mathematical models. It is hard to identify such plants by traditional identification approaches. If we have no idea about what input signals dominates the output, the modeling fuzzy system will require a lot of hidden-layer nodes in the neural networks. If we can decompose a MIMO into many simpler MISO plants, it would be very helpful to a control designer to find out the respective effect that an input would impose on an output. Using the proposed approach and evaluating the input variables based on the NZI can efficiently accomplish the task of decomposition.

5. Simulation

In this section, we apply the proposed approach to determine significant variables. First, we choose two systems to be identified:

Example 2. Consider the nonlinear function $y=(2+x_1^{1.5}-1.5\sin(3x_2))^2$. We randomly take 100 points from $0 \leqslant x_1,\ x_2 \leqslant 3$ and obtain 100 sets of input–output data. We also add a dummy variable x_3 with the same range $[0,3]$. We define three linguistic terms for each variable. All the consequence parameters are initially set to zeros and all the learning rates are defined to be 0.01. The simulation results are shown in Table 1, where $a_j=\sum_{i=1}^{n}|a_j^i|,\ j=0,1,2,\ldots,n$. From Table 1, we can find NZI_3 is much less than $\mathrm{NZI}_i,\ i=0,1,2$. Therefore, we can clearly find out the significant variables are x_1 and x_2. The simulation results are compared with [13] as tabulated in Table 2. Table 3 shows the relation between PI after 5000 iterations with four rules and data number in the simulation. Fig. 5 shows the performance index versus iterations.

Example 3. In this example, we try to model a time-delay system. Its mathematical equation is expressed as

$$y(t)=(x(t-1)+x(t-2))/2+\sin((x(t-1)+x(t-2))/2). \tag{20}$$

Assume we know the delay time of the system is less than two. So we can only take $x(t)$, $x(t-1)$, $x(t-2)$ into consideration; i.e., there are three initial input signals for the fuzzy inference engine. There are three terms for each input signal and totally 27 fuzzy rules. Table 4 shows the simulation data and the NZI for

Table 1
Near-zero index (NZI) value of each variable

Variable	NZI value
$x_0=1$	372.208835
x_1	279.786558
x_2	1081.63277
x_3	83.666059

Table 2
Model performance and training iterations for the nonlinear system $y=(2+x_1^{1.5}-1.5\sin(3x_2))^2$ using various number of rules

Number of rules	Method in [13]		Proposed method	
	Performance	Training steps	Performance	Training steps
4	0.00497	2800	0.00499	875
5	0.00336	5000	0.00337	900
6	0.00183	4400	0.00183	925
7	0.00144	5000	0.00144	1850

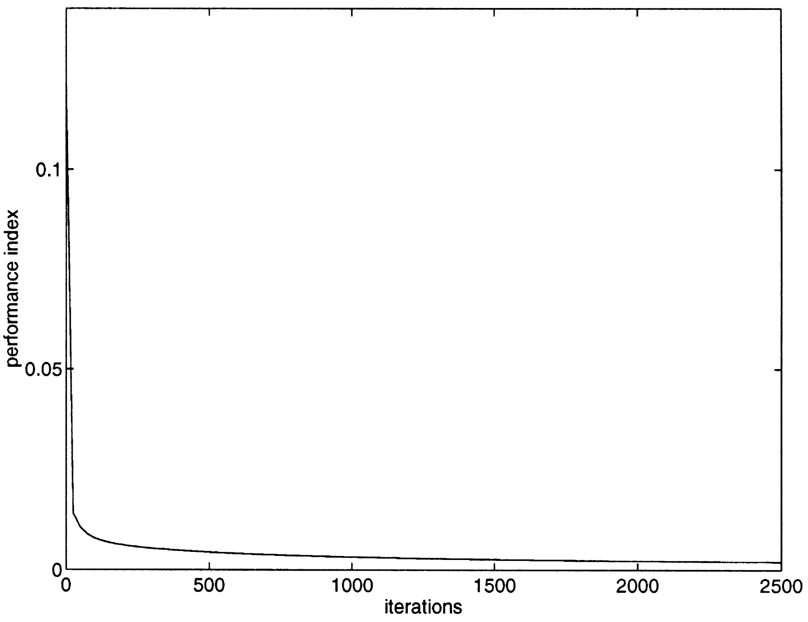

Fig. 5. Model performance index on the nonlinear system $y = (2 + x_1^{1.5} - 1.5\sin(3x_2))^2$.

Table 3
The relation between performance index values and number of data

Number of data	Performance index value
25	0.00879223
50	0.00639761
75	0.00305832
100	0.00275896
125	0.00269427
150	0.00204051
200	0.00173671
250	0.00149740
300	0.00148791

Table 4
Near-zero index (NZI) value of each variable of a time-delay system

Variable	NZI value
$x_0 = 1$	30.62493/27
$x_1 = x(t)$	4.139764/27
$x_2 = x(t-1)$	8.222736/27
$x_3 = x(t-2)$	7.977414/27

each input signal. From Table 4, we can know $x(t-1)$ and $x(t-2)$ are more important than $x(t)$, since NZI_2 and NZI_3 are larger than NZI_1.

After determining the significant variables, $x(t-1)$ and $x(t-2)$, we assign two membership functions to each variable, i.e. we use four fuzzy rules to model the system. After finishing 5000 iterations, the excellent simulation results are shown in Fig. 6.

6. Summary

For improving the fuzzy modeling precision and reduce the size of fuzzy rule base, we propose an approach for determining significant variables. The significant variables are determined based on the LDRs or Takagi–

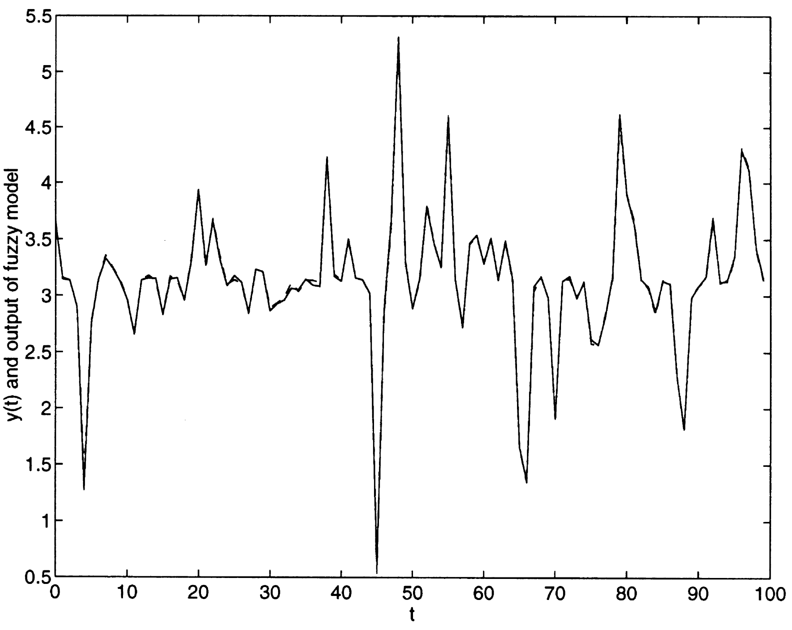

Fig. 6. Output of fuzzy model and actual output of time-delay system. The desired output is shown by the dashed line and the actual output by the solid line.

Sugeno fuzzy rules. The use of near-zero index can point out the respective importance of each input signal in the input–output relations. A significant variable is selected according to its near-zero index value. For a MIMO system, we can consider it as several MISO systems and the proposed method can be applied to efficiently realize this concept.

References

[1] F. Bouslama, A. Ichikawa, Application of neural networks to fuzzy control, Neural Networks 6 (1993) 791–799.

[2] C.-H. Chang, F.-H. Huang, J.-Y. Cheung, Design of a fuzzy controller using input and output mapping factors, IEEE Trans. Systems Man Cybernet. 21 (1991) 952–960.

[3] J. Hao, J. Vandewalle, A rule-based neural controller for inverted pendulum system, Internat. J. Neural Systems 4 (1993) 55–64.

[4] S.-Z. He, S. Tan, C.-C. Hang, Control of dynamical processes using an on-line rule-adaptive fuzzy control system, Fuzzy Sets and Systems 54 (1993) 11–22.

[5] R. Hecht-Nielsen, Neurocomputing, Addison-Wesley, Reading, MA, 1990.

[6] S.-I. Horikawa, T. Furuhashi, Y. Uchikawa, On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm, IEEE Trans. Neural Networks 3 (1992) 801–806.

[7] H. Ishibuchi, R. Fujioka, H. Tanaka, Neural networks that learn from fuzzy if–then rules, IEEE Trans. Fuzzy Systems 1 (1993) 85–97.

[8] J.S. Jang, Fuzzy modeling using generalized neural networks and Kalman filter algorithm, in: Proc. 9th National Conf. Artificial Intelligence, Anaheim, 1991, pp. 762–767.

[9] J.-S.R. Jang, Self-learning fuzzy controllers based on temporal back propagation, IEEE Trans. Neural Networks 3 (1992) 714–723.

[10] H.M. Kim, J.M. Mendel, Fuzzy basis functions: comparisons with other basis functions, IEEE Trans. Fuzzy Systems 3 (1995) 158–167.

[11] C.-C. Lee, Fuzzy logic in control systems: fuzzy logic controller – parts I and II, IEEE Trans. Systems Man Cybernet. 20 (1990) 404–435.

[12] C.-H. Lee, S.-D. Wang, A self-organizing adaptive fuzzy controller, Fuzzy Sets and Systems 80 (1996) 295–314.

[13] Y. Lin, G.A. Cunningham III, A new approach to fuzzy-neural system modeling, IEEE Trans. Fuzzy Systems 3 (1995) 190–197.

[14] C.-T. Lin, C.S.G. Lee, Neural-network-based fuzzy logic control and decision system, IEEE Trans. Comput. 40 (1991) 1320–1336.
[15] C.-T. Lin, Y.-C. Lu, A neural fuzzy system with linguistic teaching signals, IEEE Trans. Fuzzy Systems 3 (1995) 169–189.
[16] S. Mitra, S.K. Pal, Fuzzy multi-layer perceptron, inferencing and rule generation, IEEE Trans. Neural Networks 6 (1995) 51–63.
[17] R.W. Newcomb, Nonlinear Systems Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1978.
[18] D.H. Nguyen, B. Widrow, Neural networks for self-learning control systems, IEEE Control Systems Mag. 10 (1990) 18–23.
[19] J. Nie, D.A. Linkens, Neural network-based approximate reasoning: principles and implementation, Internat. J. Control 56 (1992) 399–413.
[20] J. Nie, D.A. Linkens, Learning control using fuzzified self-organizing radial basis function network, IEEE Trans. Fuzzy System 1 (1993) 280–287.
[21] R.M. Sanner, J.-J.E. Slotine, Gaussian networks for direct adaptive control, IEEE Trans. Neural Networks 3 (1992) 837–863.
[22] C.-Y. Su, Y. Stepanenko, Adaptive control of a class of nonlinear systems with fuzzy logic 2 (1994) 285–294.
[23] M. Sugeno, G.T. Kang, Structure identification of fuzzy model, Fuzzy Sets and Systems 28 (1988) 15–33.
[24] K. Sugiyama, Rule-based self-organizing controller, in: M.M. Gupta, T. Yamakawa (Eds.), Fuzzy Computing, North-Holland, Amsterdam, 1988, pp. 341–353.
[25] C.-T. Sun, Rule-base structure identification in an adaptive network based fuzzy inference system, IEEE Trans. Fuzzy Systems 2 (1994) 64–73.
[26] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Trans. System Man Cybernet. 15 (1985) 116–132.
[27] L.-X. Wang, Stable adaptive fuzzy control of nonlinear systems, IEEE Trans. Fuzzy Systems 1 (1993) 146–155.
[28] L.-X. Wang, J.M. Mendel, Fuzzy basis functions, universal approximation, and orthogonal least-squares learning, IEEE Trans. Neural Networks 3 (1992) 807–814.
[29] C.-W. Xu, Y.-Z. Lu, Fuzzy model identification and self-learning for dynamic systems, IEEE Trans. System Man Cybernet. SMC-17 (1987) 683–689.