Data-Based System Modeling Using a Type-2 Fuzzy Neural Network with a Hybrid Learning Algorithm

Chi-Yuan Yeh, Wen-Hau Roger Jeng, and Shie-Jue Lee, Member, IEEE

Abstract—We propose a novel approach for building a type-2 neural-fuzzy system from a given set of input–output training data. A self-constructing fuzzy clustering method is used to partition the training dataset into clusters through input-similarity and output-similarity tests. The membership function associated with each cluster is defined with the mean and deviation of the data points included in the cluster. Then a type-2 fuzzy Takagi-Sugeno-Kang IF-THEN rule is derived from each cluster to form a fuzzy rule base. A fuzzy neural network is constructed accordingly and the associated parameters are refined by a hybrid learning algorithm which incorporates particle swarm optimization and a least squares estimation. For a new input, a corresponding crisp output of the system is obtained by combining the inferred results of all the rules into a type-2 fuzzy set, which is then defuzzified by applying a refined type reduction algorithm. Experimental results are presented to demonstrate the effectiveness of our proposed approach.

Index Terms—Fuzzy clustering, Karnik–Mendel algorithm, least squares estimation, particle swarm optimization, type reduction, type-2 fuzzy set.

I. INTRODUCTION

THE purpose of system modeling is to model the operation of an unknown system from a given set of input–output training data. Through the simulated model, one can easily understand the underlying properties of the unknown system and handle it properly. However, this problem can be very difficult when the unknown system is highly nonlinear and complex. Many approaches have been proposed for system modeling. Quantitative approaches based on conventional mathematics tend to be more accurate, but they are not suitable when the underlying system is complex, ill defined, or uncertain. The fuzzy modeling approach was proposed to deal with this difficulty [1]. However, this approach lacks an effective learning algorithm to refine the membership functions to minimize output errors. Another approach using neural networks has been proposed [2]. This approach is capable of learning and can obtain high-precision results. However, it usually encounters problems of slow convergence and low understandability of the associated numerical weights. Recently, neural-fuzzy modeling has attracted a lot of attention [3]–[6]. This approach involves two major phases, structure identification and parameter identification, as shown in Fig. 1. Fuzzy and neural network techniques are usually adopted in the two phases. Consequently, neural-fuzzy modeling possesses the advantages of both modeling approaches. In the structure identification phase, fuzzy rules are developed from the given set of input–output training data. For the purpose of high precision, the parameters associated with the rules are then refined through learning in the parameter identification phase.

Type-1 fuzzy sets, which represent uncertainties by numbers in the range [0, 1], have been widely adopted in neural-fuzzy systems [7]. However, the membership functions of type-1 fuzzy sets are often overly precise, requiring each element of the universal set to be assigned a particular real number [8]. Type-2 fuzzy sets were proposed for dealing with this difficulty [8]. A type-2 fuzzy set allows its associated membership degrees to be uncertain and expressed as type-1 fuzzy sets [9]–[16]. The theory of type-2 fuzzy logic systems was studied in [9], [10], [12], [13], [16], and [17]. However, most of the developed type-2 fuzzy logic systems adopted interval type-2 fuzzy sets which are a special type of type-2 fuzzy sets. This may be partly due to the fact that the inference involving type-2 fuzzy sets is much more complex and time consuming than that involving interval type-2 fuzzy sets. Liang and Mendel developed interval type-2 Takagi-Sugeno-Kang (TSK) fuzzy logic systems [11], [18]. Sepúlveda et al. [19] proposed an efficient implementation of interval type-2 fuzzy logic systems. Many successful applications of interval type-2 fuzzy logic systems have been published, such as automatic control [19]–[26], function approximation [27]–[30], data classification [31]–[37], and medical diagnosis [38], [39].

To apply type-2 fuzzy systems, defuzzifying a type-2 fuzzy set is often required. Two steps are usually involved in this task. Firstly, type reduction is performed, which reduces a type-2 fuzzy set to a type-1 fuzzy set. Secondly, defuzzification is applied to the resulting type-1 fuzzy set to get the desired crisp number. Liu [40] proposed a centroid type-reduction strategy using α-cuts to decompose a type-2 fuzzy set into a collection of interval type-2 fuzzy sets, called α-planes [17], and then applying the Karnik–Mendel algorithm [41], [42] for each interval type-2 fuzzy set. Yeh et al. [43] proposed a refined type-reduction algorithm which employs the result obtained in the previous iteration to initialize the values in the current iteration. Convergence in each iteration speeds up and type reduction for type-2 fuzzy sets is done faster. Coupland and John [44] proposed a geometric-based defuzzification
method for type-2 fuzzy sets. However, it has a limitation on the form of fuzzy sets being used. Rotationally symmetrical membership functions are to be avoided in a practical system. Lucas et al. [34] calculated a centroid for each vertical slice, which is a type-1 fuzzy set. The calculated centroids are then combined to form a type-reduced set. Tan and Wu [45] proposed type-reduction strategies for type-2 fuzzy logic systems using genetic algorithms (GAs). By replacing a type-2 fuzzy set with a collection of equivalent type-1 fuzzy sets, type reduction is simplified to deciding which equivalent type-1 fuzzy set to employ in a particular situation. A GA is used for selecting the equivalent type-1 fuzzy set through an evolution process.

For structure identification, Lin et al. [46] proposed a method of fuzzy partitioning to extract initial fuzzy rules, but it is hard to decide the locations of cuts and the time complexity is high. Wong and Chen [47] proposed a clustering algorithm for the job. However, convergence is very slow especially when the amount of the training data is huge. Thawomas and Abe [48] also proposed a method for extracting fuzzy rules from a set of training patterns. To improve the approximation accuracy, the method needs to resolve the problem of overlapping among hyperboxes of different classes. Each time, two hyperboxes are considered and new hyperboxes are created for the overlapping area between them. The process iterates until no overlapping occurs between any two classes. For parameter identification, several methods for refining the parameters of type-2 fuzzy rules were proposed. Wang et al. [49] proposed a neural network for interval type-2 fuzzy rules. Gradient descent was adopted for refinement. Hagras [50] corrected some equations for it. Lee et al. [51] proposed a recurrent neural network for interval type-2 fuzzy rules using asymmetric Gaussian principal membership functions. The corresponding refining algorithm was derived from gradient descent. Juang and Tsao [52] proposed a self-evolving neural network for interval type-2 fuzzy rules using symmetric Gaussian principal membership functions and a hybrid learning algorithm which combines gradient descent and the rule-order Kalman filter. Castro et al. [53] proposed three interval type-2 fuzzy neural network architectures for time-series forecasting. Gradient descent backpropagation and gradient descent with adaptive learning rate backpropagation were adopted for refining the parameters.

We propose a novel approach for constructing a type-2 neural-fuzzy system from a given set of input–output training data. In the structure identification phase, a self-constructing fuzzy clustering method is used to partition the training dataset into clusters through input-similarity and output-similarity tests. The membership function associated with each cluster is defined with the mean and deviation of the data points included in the cluster. Then a type-2 fuzzy TSK IF-THEN rule is developed from each cluster to form a fuzzy rule base. In the parameter identification phase, a fuzzy neural network is constructed for the rules and the associated parameters are refined by a hybrid learning algorithm which incorporates particle swarm optimization (PSO) [54] and least squares estimation (LSE) [55]. For a new input, a corresponding crisp output of the system is obtained by combining the inferred results of all the rules into a type-2 fuzzy set, which is then defuzzified by applying a refined type reduction algorithm.

The rest of this paper is organized as follows. Section II introduces some basic fuzzy concepts. Section III concerns defuzzification of type-2 fuzzy sets. Fuzzy inference for type-2 fuzzy systems is outlined in Section IV. The derivation of type-2 fuzzy rules from a given set of input–output training data is described in Section V. Applying learning techniques to refine the parameters associated with the rules is described in Section VI. Experimental results are presented in Section VII. Finally, a conclusion is given in Section VIII.

II. BASIC FUZZY CONCEPTS

A type-2 fuzzy set $\tilde{A}$ on a given universal set $X$ is characterized by the membership function $\mu_{\tilde{A}}(x, u)$, where $x \in X$ and $u \in I_u \subseteq [0, 1]$, and can be represented as [41]

$$\tilde{A} = \{(x, u) \mid \mu_{\tilde{A}}(x, u) \mid \forall x \in X, \forall u \in I_u\}$$

in which $0 \leq \mu_{\tilde{A}}(x, u) \leq 1$. We refer to $\mu_{\tilde{A}}(x) = \int_{u \in I_u} \mu_{\tilde{A}}(x, u) / u$ as a secondary membership function, which is a type-1 fuzzy set. Obviously, $\tilde{A}$ can also be represented as $\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid \forall x \in X\}$. If $\mu_{\tilde{A}}(x)$ is an interval for all $x \in X$, then $\tilde{A}$ is an interval type-2 fuzzy set. Furthermore, if $\mu_{\tilde{A}}(x)$ is a number in the range $[0, 1]$, $\tilde{A}$ reduces to a type-1 fuzzy set. Also, the principal membership function of $\tilde{A}$ is defined to consist of all the points whose secondary membership degree is equal to one [12], [31].

When $X$ is discretized into $n$ points, $x_1, x_2, \ldots, x_n$, (2) becomes

$$\tilde{A} = \sum_{i=1}^{n} \left[ \int_{u \in I_u} \frac{\mu_{\tilde{A}}(x_i, u)}{u} \right] x_i$$

and

$$= \sum_{i=1}^{n} \frac{\mu_{\tilde{A}}(x_i)}{x_i}.$$  

The centroid of $\tilde{A}$ can then be defined as follows [41]:

$$C(\tilde{A}) = \int_{u_1 \in I_u} \cdots \int_{u_n \in I_u} \left[ \mu_{\tilde{A}}(x_1, u_1) \star \cdots \star \mu_{\tilde{A}}(x_n, u_n) \right]$$

where $\star$ is the minimum $t$-norm operator. Note that, if $\tilde{A}$ is a type-1 fuzzy set, $C(\tilde{A})$ is a scalar [7]. If $\tilde{A}$ is an interval type-2 fuzzy set, $C(\tilde{A})$ is an interval set [41], [42], [56], [57]. If $\tilde{A}$ is a type-2 fuzzy set, $C(\tilde{A})$ is a type-1 fuzzy set [40], [41], [44].
function form and upper membership functions for the FOU are
\( \tilde{\mu}_A(x) = \text{gauss}(u; \text{gauss}(x; m^p, \sigma^p), \sigma^s) \) (6)
where \( x \in X, u \in [0, 1], \) and \( \text{gauss}(x; m, \sigma) \) is the Gaussian function with mean \( m \) and deviation \( \sigma \) defined as
\[
\text{gauss}(x; m, \sigma) = \exp \left[ -\frac{(x - m)^2}{\sigma^2} \right].
\] (7)
The \( \alpha \)-cut of (6) is
\[
[u|\mu_A(x) \geq \alpha \wedge u \in [0, 1]].
\]
Obviously, the inner Gaussian \( \text{gauss}(x; m^p, \sigma^p) \) is the principal membership function for \( A \). Fig. 2 shows such a type-2 fuzzy set with \( m^p = 5 \) and \( \sigma^p = 2 \), in which the principal membership function is indicated as a thick line. For \( x = 7, \)
\( \text{gauss}(7; m^p, \sigma^p) = e^{-1}. \) Furthermore, let \( \sigma^s = 0.2. \) Then the secondary membership function \( \mu_A(7) \) is
\[
\mu_A(7) = \text{gauss}(u; \text{gauss}(7; m^p, \sigma^p), \sigma^s)
\] = \[
\exp \left[ -\frac{(u - e^{-1})^2}{0.2} \right]
\]
which, also shown in the figure, is a type-1 fuzzy set. Note that the domain of the secondary membership functions \( J_k \)
is restricted to \([0, 1]\). Therefore, the footprint of uncertainty (FOU) of our type-2 fuzzy sets consists of all the points \((x, u)\)
in the 2-D space such that \( x \in X \) and \( u \in [0, 1]. \) The lower and upper membership functions for the FOU are \( u = 0 \) and \( u = 1 \), respectively.

### III. Defuzzification of Type-2 Fuzzy Sets

The result after inference of a type-2 fuzzy system is usually a type-2 fuzzy set. For practical applications, the output is required to be a crisp number. Defuzzifying a type-2 fuzzy set is usually done in two steps. Firstly, we compute the centroid of the type-2 fuzzy set which is a type-1 fuzzy set. Secondly, the centroid of the resulting type-1 fuzzy set is computed and the desired crisp number is obtained.

The algorithm proposed in [43] is adopted for type reduction. This algorithm is a refinement to Liu’s algorithm [40]. Both algorithms are based on the Karnik–Mendel algorithm [41], [42], which is an efficient method for computing the centroid of an interval type-2 fuzzy set. Suppose we are given a type-2 fuzzy set \( \tilde{A} \) on a universal set \( X = \{x_1, x_2, \ldots, x_n\} \), where \( x_1 < x_2 < \cdots < x_n \). Let \( \alpha_1, \alpha_2, \ldots, \alpha_k \) be \( k \) real numbers, and \( \alpha_s \in [0, 1], 1 \leq s \leq k \). For each \( \alpha_s \), we take \( \alpha_s \)-cuts of the secondary membership functions of \( \tilde{A} \) at \( x_1, x_2, \ldots, x_n \) and let them be intervals \( [L_{s,1}, T_{s,1}], [L_{s,2}, T_{s,2}], \ldots, [L_{s,n}, T_{s,n}] \). Group these \( n \) intervals and we have an interval type-2 fuzzy set \( \alpha^s \tilde{A} \) on \( X \) for \( \alpha_s \). As mentioned in Section II, the centroid of each \( \alpha^s \tilde{A} \) is an interval. Let the centroid of \( \alpha^s \tilde{A} \) be the interval \([\bar{b}_s, \bar{b}_s]\). Then, \([\bar{b}_s, \bar{b}_s], 1 \leq s \leq k \), can be computed efficiently as follows.

**Step 1:** Finding \( \bar{b}_s, s = k, k - 1, \ldots, 1 \).

1) Initialization.
   a) If \( s = k \), then
   \[
   \bar{b}_s = \frac{\sum_{j=1}^{n} x_j \left( \frac{L_{s,j+1} + T_{s,j}}{2} \right)}{\sum_{j=1}^{n} \left( \frac{L_{s,j+1} + T_{s,j}}{2} \right)}.
   \] (8)
   Also, set \( L_{s,j} = i \) such that \( x_i \leq \bar{b}_s < x_{i+1} \).
   b) If \( s \neq k \), then set \( L_{s,j} = L_{s+1,j+1} \) only.

2) Update. Update \( \bar{b}_s \) by
   \[
   \bar{b}_s = \frac{\sum_{j=1}^{n} x_j T_{s,j} + \sum_{j=L_{s,j+1}}^{n} x_j L_{s,j}}{\sum_{j=1}^{n} T_{s,j} + \sum_{j=L_{s,j+1}}^{n} L_{s,j}}
   \] (9)
   and \( L_{s,j} \) such that
   \[
   x_{L_{s,j}} \leq \bar{b}_s < x_{L_{s,j+1}}.
   \] (10)

If the value of \( L_{s,j} \) changes, we continue to update \( \bar{b}_s \) and \( L_{s,j} \) by (9) and (10). Otherwise, we are done with \( \bar{b}_s \) and \( L_{s,j} \), and \( L_{s,j} \) is called the left switch point of \( \alpha^s \tilde{A} \).

**Step 2:** Finding \( \overline{b}_s, s = k, k - 1, \ldots, 1 \).

1) Initialization.
   a) If \( s = k \), then
   \[
   \overline{b}_s = \frac{\sum_{j=1}^{n} x_j \left( \frac{L_{s,j+1} + T_{s,j}}{2} \right)}{\sum_{j=1}^{n} \left( \frac{L_{s,j+1} + T_{s,j}}{2} \right)}.
   \] (11)
   Also, set \( \overline{L}_{s,j} = i \) such that \( x_i \leq \overline{b}_s < x_{i+1} \).
   b) If \( s \neq k \), then set \( \overline{L}_{s,j} = \overline{L}_{s+1,j+1} \) only.

2) Update. Update \( \overline{b}_s \) by
   \[
   \overline{b}_s = \frac{\sum_{j=1}^{n} x_j L_{s,j} + \sum_{j=L_{s,j+1}}^{n} x_j T_{s,j}}{\sum_{j=1}^{n} L_{s,j} + \sum_{j=L_{s,j+1}}^{n} T_{s,j}}
   \] (12)
and $\mathbf{s}$ such that
\[ x_{T_s} \leq \bar{b}_s < x_{T_s+1}. \] (13)

If the value of $\mathbf{s}$ changes, we continue to update $\bar{b}_s$ and $\mathbf{s}$ by (12) and (13). Otherwise, we are done with $\bar{b}_s$ and $\mathbf{s}$, and $\mathbf{s}$ is called the right switch point of $a_i \mathbf{A}$.

Obviously, (9) and (12) hold for the centroid $[p_1, \bar{b}_s]$, the left switch point, and each rule $a_i \mathbf{A}$, $1 \leq s \leq k$. When all the centroids of $a_i \mathbf{A}$ are computed, the centroid of $\mathbf{A}$ can then be derived as
\[ C(\mathbf{A}) \approx \frac{\sum_{s=1}^{k} \alpha_s}{\sum_{s=1}^{k} \alpha_s (p_s + \bar{b}_s)} \] (14)
by the first decomposition theorem of type-1 fuzzy sets [7].

Note that (14) is a type-1 fuzzy set. To get a crisp defuzzi-fied value $y$ as the system output, we approximate the centroid of (14)
\[ y = \frac{\int C(\mathbf{A}) dx}{\int C(\mathbf{A}) dx} \]
by [58]
\[ y \approx \frac{\sum_{s=1}^{k} \alpha_s \cdot (p_s + \bar{b}_s)}{2 \sum_{s=1}^{k} \alpha_s} = \frac{\sum_{s=1}^{k} \alpha_s (p_s + \bar{b}_s)}{2 \sum_{s=1}^{k} \alpha_s}. \] (15)
Using (15) instead of a mathematically correct centroid allows us to apply LSE in the parameter identification phase, as described in Section VI-B.

IV. TYPE-2 FUZZY INFERENCE

Let $x_1, x_2, \ldots, x_n$ be the input variables and $y$ be the output variable. A type-2 fuzzy system we are concerned with this paper contains a rule base of $J$ rules, $R_1, R_2, \ldots, R_J$, and each rule $R_j$, $1 \leq j \leq J$, takes the following TSK-based form [11, 18]:
\[ R_j: \text{IF } x_1 \text{ IS } \mathbf{A}_{1,j} \text{ AND } x_2 \text{ IS } \mathbf{A}_{2,j} \text{ AND } \cdots \text{ AND } x_n \text{ IS } \mathbf{A}_{n,j} \text{ THEN } y \text{ IS } c_j = \beta_{0,j} + \beta_{1,j} x_1 + \cdots + \beta_{n,j} x_n \] (16)
where $\beta_{0,j}, \beta_{1,j}, \ldots, \beta_{n,j}$ are real-valued parameters and $\mathbf{A}_{1,j}, \mathbf{A}_{2,j}, \ldots, \mathbf{A}_{n,j}$ are type-2 fuzzy sets for $x_1, x_2, \ldots, x_n$, respectively.

Based on the $J$ rules, a four-layer fuzzy neural network can be constructed as shown in Fig. 3. For a set of input values, $\mathbf{x} = [p_1, p_2, \ldots, p_n]$, a crisp output $y$ can be obtained through the operation of the fuzzy neural network as follows.

1) Layer 1 contains $J$ groups. Group $j$ corresponds to rule $R_j$ and contains $n$ nodes. Node $i$ of group $j$ produces its output, $o^{(1)}_{i,j}$, by computing the value of the corresponding membership function
\[ o^{(1)}_{i,j} = \mu_{\mathbf{A}_{i,j}}(p_i) \] (17)
for $1 \leq i \leq n$ and $1 \leq j \leq J$. Note that each $o^{(1)}_{i,j}$ is a type-1 fuzzy set.

2) Layer 2 contains $J$ nodes. Node $j$ produces its output, $o^{(2)}_j$, by computing the firing strength of rule $R_j$
\[ o^{(2)}_j = o^{(1)}_{1,j} \cdot o^{(1)}_{2,j} \cdots o^{(1)}_{n,j} \] (18)
for $1 \leq j \leq J$, where we adopt the arithmetic product “$\cdot$” for “AND.” For two type-1 fuzzy sets $A$ and $B$, $A \times B$ is defined as [7]
\[ A \times B = \bigcup_{a \in [0,1]} \frac{\alpha}{\alpha} A \times B \] (19)
\[ \alpha(A \times B) = [\alpha A \times \alpha B] \] (20)
where $A$ and $B$ are $\alpha$-cuts of $A$ and $B$, respectively, $A_{\alpha}$ and $B_{\alpha}$ denote the lower bound of $\alpha A$ and $\alpha B$, respectively, and $A_{\alpha}$ and $B_{\alpha}$ denote the upper bound of $\alpha A$ and $\alpha B$, respectively. The product operation in (19) produces the same result as the more commonly used meet operation [13]. However, (19) involves less computation. Besides, the way using $\alpha$-cuts matches the way type reduction of a type-2 fuzzy set is done. Note that $A \times B$ is also a type-1 fuzzy set. Obviously, each $o^{(2)}_j$ is a type-1 fuzzy set.

3) Layer 3 contains only one node whose output, $o^{(3)}$, is the result of performing type reduction on its input. In this layer, the consequent value $c_j$ of each rule $R_j$ is computed as
\[ c_j = \beta_{0,j} + \beta_{1,j} p_1 + \cdots + \beta_{n,j} p_n \] (21)
for $1 \leq j \leq J$. Next, $c_1, c_2, \ldots, c_J$ are sorted in ascending order, and let the resulting sequence be $c_{j_1}, c_{j_2}, \ldots, c_{j_J}$. Obviously, the set
\[ \bar{c} = \sum_{i=1}^{J} o^{(2)}_i \] (22)
is a type-2 fuzzy set defined on the discrete universal set $\{c_{j_1}, c_{j_2}, \ldots, c_{j_J}\}$. Then type reduction, as presented
in Section III, is carried out on \( D \) and \( o^{(3)} \) becomes the result of (14).

4) Layer 4 contains only one node which performs defuzzification on its input, \( o^{(3)} \). Its output, \( o^{(4)} \), is the result of (15).

The crisp system output \( y \) corresponding to the input \( x = [p_1, p_2, \ldots, p_n] \) is then set to be the output \( o^{(4)} \).

V. Structure Identification

As usual, our approach to system modeling based on a given set of input–output training data consists of two stages, structure identification and parameter identification. In the structure identification phase, as shown in Fig. 4, initial fuzzy rules are developed from the given set of input–output training data. First, a self-constructing fuzzy clustering algorithm is employed to partition the training data into a collection of fuzzy clusters. Second, the clusters are converted into a rule base of type-2 fuzzy TSK IF-THEN rules. For the purpose of high precision, the parameters associated with the rules are then refined through learning in the parameter identification phase, to be presented later.

Assume the system to be modeled has \( n \) inputs, \( x_1, x_2, \ldots, x_n \) and one output \( y \). The given training dataset contains \( \ell \) patterns \(<\mathbf{p}^{(1)}, \mathbf{d}^{(1)}>\), \(<\mathbf{p}^{(2)}, \mathbf{d}^{(2)}>, \ldots, <\mathbf{p}^{(\ell)}, \mathbf{d}^{(\ell)}>\) where \( \mathbf{p}^{(r)} = [p^{(r)}_1, p^{(r)}_2, \ldots, p^{(r)}_n] \) denotes the \( n \) input values and \( \mathbf{d}^{(r)} \) denotes the corresponding desired output value of the \( r \)th pattern, \( 1 \leq r \leq \ell \). The self-constructing fuzzy clustering method partions the training dataset into clusters through input-similarity and output-similarity tests. A cluster contains a certain number of training patterns, and is characterized by the product of \( n \) 1-D Gaussian functions. Gaussian functions are adopted because of their superiority over other functions in performance [59], [60]. In addition, each cluster keeps a height which is the average of the desired outputs of the patterns contained in the cluster. The clustering algorithm is an incremental self-constructing learning approach. Training patterns are considered one by one. No clusters exist at the beginning, and clusters can be created if necessary. For each pattern, the similarity of this pattern to each existing cluster is calculated to decide whether it is combined into an existing cluster or a new cluster is created. When all training patterns have been considered, we get a number of clusters that are then converted to type-2 fuzzy TSK IF-THEN rules. Details are described below.

Let \( k \) be the number of currently existing clusters, denoted by \( G_1, G_2, \ldots, G_k \), respectively. Each cluster \( G_j \) has mean \( \mathbf{m}_j = [m_{j,1}, m_{j,2}, \ldots, m_{j,n}] \), deviation \( \mathbf{\sigma}_j = [\sigma_{j,1}, \sigma_{j,2}, \ldots, \sigma_{j,n}] \), and height \( h_j \). Let \( S_j \) be the size of cluster \( G_j \), i.e., the number of training patterns contained in \( G_j \). Initially, we have \( k = 0 \). So, no clusters exist at the beginning. For pattern \( r \), i.e., \(<\mathbf{p}^{(r)}, \mathbf{d}^{(r)}>\), \( 1 \leq r \leq \ell \), we calculate the input similarity of pattern \( r \) to each existing cluster \( G_j \) as follows:

\[
\mu_{G_j} (\mathbf{p}^{(r)}) = \prod_{i=1}^{n} \exp \left[ -\frac{(p^{(r)}_i - m_{i,j})^2}{\sigma_{i,j}} \right] \tag{23}
\]

for \( 1 \leq j \leq k \). Notice that \( 0 \leq \mu_{G_j} (\mathbf{p}^{(r)}) \leq 1 \). If \( \mathbf{p}^{(r)} \) is close to the mean \( \mathbf{m}_j \) of cluster \( G_j \), then \( \mu_{G_j} (\mathbf{p}^{(r)}) \approx 1 \). On the contrary, if \( \mathbf{p}^{(r)} \) is far away from the mean \( \mathbf{m}_j \) of cluster \( G_j \), then \( \mu_{G_j} (\mathbf{p}^{(r)}) \approx 0 \). We say that pattern \( r \) passes the input similarity test on cluster \( G_j \) if

\[
\mu_{G_j} (\mathbf{p}^{(r)}) \geq \rho \tag{24}
\]

where \( \rho, 0 \leq \rho \leq 1 \), is a predefined threshold. If the user intends to have larger clusters, then he/she can give a smaller threshold. Otherwise, a bigger threshold can be given. As the threshold increases, the number of clusters also increases. Furthermore, we calculate the output similarity of pattern \( r \) to each existing cluster \( G_j \) as follows:

\[
e_j = |d^{(r)} - h_j| \tag{25}
\]

where \( h_j \) is the height of \( G_j \). Let \( d_{\text{max}} \) and \( d_{\text{min}} \) denote the maximum and minimum of the desired outputs

\[
d_{\text{max}} = \max_{1 \leq \ell \leq \ell} d^{(\ell)}, \quad d_{\text{min}} = \min_{1 \leq \ell \leq \ell} d^{(\ell)}. \tag{26}
\]

We say that pattern \( r \) passes the output-similarity test on cluster \( G_j \) if

\[
e_j \leq \tau (d_{\text{max}} - d_{\text{min}}) \tag{26}
\]

where \( \tau, 0 \leq \tau \leq 1 \), is another predefined threshold. As \( \tau \) decreases, the test gets tighter and more clusters will be produced.

Two cases may occur. First, there are no existing clusters on which pattern \( r \) has passed both the input-similarity test and the output-similarity test. In this case, we assume that pattern \( r \) is not similar enough to any existing cluster. We increase \( k \) by 1. A new cluster is created by setting its mean to be \( \mathbf{p}^{(r)} \) itself, its deviation to be a default vector \( \mathbf{\sigma}_0 = [\sigma_0, \ldots, \sigma_0] \), and its height to be \( d^{(r)} \)

\[
k = k + 1, \quad \mathbf{m}_k = \mathbf{p}^{(r)}, \quad \mathbf{\sigma}_k = \mathbf{\sigma}_0, \quad h_k = d^{(r)}. \tag{27}
\]

Note that the deviation of this new cluster \( G_k \) is 0 since it contains only one member. We cannot use zero deviation in the calculation of fuzzy similarities. Therefore, we initialize the deviation \( \mathbf{\sigma}_k \) of \( G_k \) to be \( \mathbf{\sigma}_0 \). On the other hand, if there are existing fuzzy clusters on which pattern \( r \) has passed both the input-similarity test and the output-similarity test, let \( G_{j_1}, G_{j_2}, \ldots, G_{j_l} \) be such clusters and let the cluster with the greatest input similarity be cluster \( t \)

\[
t = \arg \max_{j = j_1, \ldots, j_l} \mu_{G_j} (\mathbf{p}^{(r)}). \tag{28}
\]

In this case, we assume that pattern \( r \) is closest to cluster \( G_t \). Now the mean \( \mathbf{m}_t \), the deviation \( \mathbf{\sigma}_r \), the height \( h_t \), and the size \( S_t \) of \( G_t \) are modified to include pattern \( r \) as its member. Let the superscripts \( o \) and \( n \) denote the values before and after
Algorithm

procedure Self-Constructing-Clustering
for \( r = 1 \) to \( \ell \)
  Calculate the input and output similarities between
  pattern \( r \) and each currently existing cluster by
  (23) and (25);
  if there is no cluster on which pattern \( r \) has
  passed both the input- and output-similarity
tests of (24) and (26)
    Create a new cluster by (27);
  else
    Combine pattern \( r \) to the cluster to which pattern
    \( r \) is most similar by (29)–(33);
endfor
endprocedure

modification. For example, \( S^p \) and \( S^n \) indicate the size of \( G_t \)
before and after, respectively, \( p^{(r)} \) is included. Then we have
\[
m_{i,t}^n = \frac{\sum_{i=1}^n p_i^{(o)} m_{i,t}^p + p_j^{(r)}}{S^p_t + 1} \quad \text{(29)}
\]
\[
\sigma_{i,t}^n = \frac{\sum_{i=1}^n (p_i^{(o)} - m_{i,t}^p)^2}{S^p_t - 1} \quad \text{(30)}
\]
\[
h_{i,t}^n = \frac{\sum_{i=1}^n d^{(o)} + d^{(r)}}{S^p_t + 1} \quad \text{(31)}
\]
for \( 1 \leq i \leq n \), where \( A = ((S^p_t - 1)(\sigma_{i,t}^n)^2 + S^n_t(\sigma_{i,t}^n)^2 + \left(p_j^{(r)}\right)^2)/S^p_t \) and \( B = (S^p_t/S^n_t(m_{i,t}^p)^2) \), and
\[
S^p_t = S^p_t + 1 \quad \text{(33)}
\]
which increases the size of \( G_t \) by 1. The aforementioned
process is iterated until all the training patterns have been
processed. The algorithm can be summarized as shown above.

Note that the order in which the training patterns are fed
influences the clusters obtained. We apply a heuristic to
determine the order. We sort all the patterns, in decreasing
order, by their largest input values. Then the patterns are
fed in this order. For example, let \(< 0.1, 0.3, 0.6, 0.7 \>,
\(< 0.3, 0.3, 0.4 \>, 0.9 \>, and \(< 0.8, 0.1, 0.1 \>, 0.5 \>
be three training patterns. The largest input values in these patterns
are 0.6, 0.4, and 0.8, respectively. The sorted list is 0.8, 0.6, 0.4.
So the order of feeding is pattern 3, pattern 1, and pattern 2.
In this way, more significant patterns will be fed in first
and likely become the core of the underlying cluster.

Suppose we have \( J \) clusters after clustering. Then we
convert each cluster to a type-2 fuzzy rule and we have a
set of \( J \) rules, \( R_1, R_2, \ldots, R_J \). Each rule \( R_j \) is in the form of
(16), where the membership function of \( A_{i,j} \), \( 1 \leq i \leq n \), is
\[
\mu_{A_{i,j}} (x_i) = \text{gauss}(u; \text{gauss}(x_i; m_{i,j}^p, \sigma_{i,j}^p), \sigma_{i,j}^x) \quad \text{(34)}
\]
where \( u \in [0, 1] \). The antecedent parameters of rule \( R_j \) include
\( m_{i,j}^p, \sigma_{i,j}^p, \text{ and } \sigma_{i,j}^x \), \( 1 \leq i \leq n \), which are initialized according
to the mean and deviation of cluster \( G_j \) as follows:
\[
m_{i,j}^p = m_{i,j} \quad \text{(35)}
\]
\[
\sigma_{i,j}^p = \sigma_{i,j} \quad \text{(36)}
\]
\[
\sigma_{i,j}^x = \kappa \sigma_{i,j} \quad \text{(37)}
\]
for \( 1 \leq i \leq n \), where \( \kappa \) is a small user-defined constant,
e.g., \( \kappa = 0.1 \). The consequent parameters of rule \( R_j \) include
\( \beta_{0,j}, \beta_{1,j}, \ldots, \beta_{n,j} \) which are initialized as follows. Let the \( S_j \)
training patterns included in cluster \( G_j \) be \(< p_j^{(1)}, d^{(j1)} \>,
< p_j^{(2)}, d^{(j2)} \>, \ldots, < p_j^{(J)}, d^{(jJ)} \>). Then \( \beta_{0,j}, \beta_{1,j}, \ldots,
\beta_{n,j} \) are set to be the least squares solution of
\[
\min_{\beta} \|Y - X\beta\| \quad \text{(38)}
\]
where
\[
Y = \begin{bmatrix} d^{(j1)} \ d^{(j2)} \ldots \ d^{(jJ)} \end{bmatrix}^T
\]
\[
X = \begin{bmatrix} 1 \ p_1^{(j1)} \ p_2^{(j1)} \ldots \ p_n^{(j1)} \\
1 \ p_1^{(j2)} \ p_2^{(j2)} \ldots \ p_n^{(j2)} \\
\vdots \ \vdots \ \vdots \ \vdots \\
1 \ p_1^{(jJ)} \ p_2^{(jJ)} \ldots \ p_n^{(jJ)} \end{bmatrix}
\]
\[
\beta = [\beta_{0,j} \ \beta_{1,j} \ \beta_{2,j} \ \ldots \ \beta_{n,j}]^T. \quad \text{(40)}
\]
An iterative divide-and-merge-based least squares estimator
[55], designed for solving large problems efficiently, is adopted
for finding an optimal \( \beta \) in (38).

VI. PARAMETER IDENTIFICATION

After the set of \( J \) initial fuzzy rules is obtained in the
structure identification phase, we proceed to improve the
precision of these rules, by refining the antecedent and
consequent parameters involved, through the application of
a hybrid learning algorithm. Since we have \( J \) rules and
each rule has \( 3n \) antecedent parameters, we have \( 3nJ \) such
parameters in total, i.e., \( m_{1,1}^p, \sigma_{1,1}^p, \sigma_{1,1}^x, \ldots, m_{n,1}^p, \sigma_{n,1}^p, \sigma_{n,1}^x, \ldots,
m_{1,J}^p, \sigma_{1,J}^p, \sigma_{1,J}^x, \ldots, m_{n,J}^p, \sigma_{n,J}^p, \sigma_{n,J}^x \). Also, each rule
has \( n + 1 \) consequent parameters, so we have \((n + 1)J \) such
parameters in total, i.e., \( \beta_{0,1}, \beta_{1,1}, \ldots, \beta_{n,1}, \beta_{0,1}, \beta_{1,1}, \ldots,
\beta_{n,1} \). In PSO is used to optimize the antecedent parameters
and LSE is used to optimize the consequent parameters. An
iteration of learning involves the presentation of all training
patterns. In each iteration of learning, both PSO and LSE are
applied. We first treat all the consequent parameters as fixed
and use PSO to refine the antecedent parameters. Then we
treat all the antecedent parameters as fixed and use LSE to
refine the consequent parameters. The process is iterated, as
shown in Fig. 5, until the desired approximation precision is
achieved.

A. Refining Antecedent Parameters

As mentioned earlier, we treat all the consequent para-
parameters as fixed and use PSO to refine the antecedent para-
parameters. PSO is a population-based global search algorithm
for problem solving, proposed by Kennedy and Eberhart in 1995 [54]. Given a problem, it starts with a swarm of \( S \) particles. Each particle is associated with a position, which is a candidate solution, and a velocity within the search space. In each iteration, the positions of all particles are updated. Let the position and velocity of particle \( i \) in iteration \( t \) be denoted by \( \mathbf{P}_t^{(i)} \) and \( \mathbf{V}_t^{(i)} \), respectively. In iteration \( t+1 \), we update the velocity of particle \( i \), \( 1 \leq i \leq S \), as follows:

\[
\mathbf{V}_{t+1}^{(i)} = w \times \mathbf{V}_t^{(i)} + k_1 \times \text{rand}_1 \times (\mathbf{P}_t^{(i)} - \mathbf{P}_t^{(i)}) + k_2 \times \text{rand}_2 \times (\mathbf{G}_t^{(i)} - \mathbf{P}_t^{(i)})
\]  

(41)

where \( w, k_1, \) and \( k_2 \) are the coefficients of inertia, cognitive, and social, respectively, \( \text{rand}_1 \) and \( \text{rand}_2 \) are uniformly distributed random numbers in \([0,1]\). \( \mathbf{P}_t^{(i)} \) is the best previous position of particle \( i \), and \( \mathbf{G}_t^{(i)} \) is the overall best particle position so far. Then the position of particle \( i \) is updated by

\[
\mathbf{P}_{t+1}^{(i)} = \mathbf{P}_t^{(i)} + \mathbf{V}_{t+1}^{(i)}
\]  

(42)

for \( 1 \leq i \leq S \). When all the particles in the swarm have got their positions updated, the current iteration is done and the swarm is ready for the next iteration. For more details about PSO, refer to [54].

In our case, the objective function to be optimized by PSO is the mean square error (MSE) with respect to the training patterns. The position of particle \( i \) in iteration \( t \) consists of the \( 3nJ \) antecedent parameters as follows:

\[
\mathbf{P}_t^{(i)} = \left[ m_{1,1}^p, \sigma_1^p, \mu_{1,1}, \ldots, m_{1,1}^p, \sigma_1^p, \mu_{1,1}, \ldots, m_{n,1}^p, \sigma_n^p, \mu_{n,1}, \ldots, m_{1,j}^p, \sigma_1^p, \mu_{1,j}, \ldots, m_{n,j}^p, \sigma_n^p, \mu_{n,j}, \ldots, m_{1,j}^s, \sigma_1^s, \mu_{1,j}, \ldots, m_{n,j}^s, \sigma_n^s, \mu_{n,j}, \ldots \right].
\]  

(43)

B. Refining Consequent Parameters

To refine consequent parameters, we treat all antecedent parameters as fixed (Fig. 3). For training pattern \( r \), i.e., \((\mathbf{p}^{(r)}, \mathbf{d}^{(r)})\), \( 1 \leq r \leq \ell \), let \( o_{1,j}^{(1)}(r) \), \( o_{1,j}^{(2)}(r) \), and \( o_{1,j}^{(4)}(r) \) denote the actual output of layers 1, 2, and 4, respectively, for this pattern. We have

\[
o_{1,j}^{(1)}(r) = \mu_{A_{ij}}(\mathbf{p}^{(r)})
\]  

\[
= \text{gauss}(u; \text{gauss}(\mathbf{p}^{(r)}; m_{1,j}^p, \sigma_1^s)), \sigma_{1,j}^s (44)
\]

\[
o_{1,j}^{(2)}(r) = o_{1,j}^{(1)}(r) \times o_{2,j}^{(1)}(r) \times \cdots \times o_{n,j}^{(1)}(r). \]  

(45)

By (15), we have

\[
o_{1,j}^{(4)}(r) = \frac{\sum_{s=1}^{k} \alpha_s (b_s^{(r)} + \bar{b}_s^{(r)})}{2 \sum_{s=1}^{k} \alpha_s}
\]  

(46)

where, by (9) and (12)

\[
p_r^{(r)} = \frac{\sum_{j=1}^{J} c_j^{(r)} \mathbf{T}_{j,s}^{(r)} + \sum_{j=L_{j,s}^{(r)}+1}^{J} c_j^{(r)} \mathbf{L}_{j,s}^{(r)}}{\sum_{j=1}^{J} c_j^{(r)} \mathbf{T}_{j,s}^{(r)} + \sum_{j=L_{j,s}^{(r)}+1}^{J} c_j^{(r)} \mathbf{L}_{j,s}^{(r)}}
\]  

(47)

\[
\mathbf{T}_{j,s}^{(r)} = \sum_{j=1}^{J} c_j^{(r)} \mathbf{T}_{j,s}^{(r)} + \sum_{j=L_{j,s}^{(r)}+1}^{J} c_j^{(r)} \mathbf{L}_{j,s}^{(r)}
\]  

(48)

with \( c_j^{(r)} = \sum_{s=0}^{n} \bar{R}_{i,j} \mathbf{p}_i^{(r)} \) being the consequent part of rule \( R_j \) for \( \mathbf{p}^{(r)} \). For convenience, we add the component \( p_0^{(r)} \) and set \( p_0^{(r)} = 1 \) for \( 1 \leq r \leq \ell \). Note that \( L_{j,s}^{(r)} \) is the \( \alpha_s \)-cut of \( o_{2,j}^{(1)}(r) \) for \( 1 \leq r \leq \ell, 1 \leq j \leq J, \) and \( 1 \leq s \leq k \). Let

\[
H_{i,j,s}^{(r)} = \begin{cases} p_i^{(r)} \mathbf{T}_{j,s}^{(r)}, & \text{if } j \leq L_{j,s}^{(r)} \\ p_i^{(r)} \mathbf{L}_{j,s}^{(r)}, & \text{if } j > L_{j,s}^{(r)} \end{cases}
\]  

(49)

\[
K_{i,j,s}^{(r)} = \begin{cases} p_i^{(r)} \mathbf{T}_{j,s}^{(r)}, & \text{if } j \leq L_{j,s}^{(r)} \\ p_i^{(r)} \mathbf{L}_{j,s}^{(r)}, & \text{if } j > L_{j,s}^{(r)} \end{cases}
\]  

(50)

\[
W_{i,j,s}^{(r)} = \sum_{j=1}^{L_{j,s}^{(r)}} \mathbf{T}_{j,s}^{(r)} + \sum_{j=L_{j,s}^{(r)}+1}^{J} \mathbf{L}_{j,s}^{(r)}
\]  

(51)

\[
Z_{i,s}^{(r)} = \sum_{j=1}^{L_{j,s}^{(r)}} \mathbf{T}_{j,s}^{(r)} + \sum_{j=L_{j,s}^{(r)}+1}^{J} \mathbf{L}_{j,s}^{(r)}
\]  

(52)
Then (46) becomes
\[
\phi^{(4)}(r) = \sum_{s=1}^{k} \alpha_s \left( \sum_{j=1}^{J} \alpha_j \left( \frac{H^{(r)}_{s,j} + K^{(r)}_{s,j}}{W^{(s)}} \right) \right) \beta_{t,j} \\
= \sum_{s=1}^{k} \alpha_s \left( \sum_{j=1}^{J} \alpha_j \left( \frac{H^{(r)}_{s,j} + K^{(r)}_{s,j}}{W^{(s)}} \right) \right) \beta_{t,j}
\]
Let
\[
x^{(r)}_{t,j} = \sum_{s=1}^{k} \alpha_s \left( \frac{H^{(r)}_{s,j} + K^{(r)}_{s,j}}{W^{(s)}} \right)
\]
We have
\[
\phi^{(4)}(r) = \sum_{j=1}^{J} \sum_{t=0}^{\ell} x^{(r)}_{t,j} \beta_{t,j}
\]
for \(1 \leq r \leq \ell\). Letting \(d^{(r)} = \phi^{(4)}(r)\), we have
\[
d^{(r)} = \sum_{j=1}^{J} \sum_{t=0}^{\ell} x^{(r)}_{t,j} \beta_{t,j}
\]
for \(1 \leq r \leq \ell\), resulting in the following linear system:
\[
Y = X\beta
\]
where
\[
Y = \begin{bmatrix} d^{(1)}(1) & d^{(2)}(1) & \cdots & d^{(\ell)}(1) \\
(1) & x^{(1)}_{0,1} & x^{(1)}_{0,2} & \cdots & x^{(1)}_{0,J} \\
(2) & x^{(2)}_{1,1} & x^{(2)}_{1,2} & \cdots & x^{(2)}_{1,J} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
J & x^{(J)}_{\ell,1} & x^{(J)}_{\ell,2} & \cdots & x^{(J)}_{\ell,J} \\
\end{bmatrix}^T \\
X = \begin{bmatrix} \beta_0, \beta_{1,1}, \beta_{1,2}, \cdots, \beta_{1,J} \\
1, \beta_{0,1}, \beta_{0,2}, \cdots, \beta_{0,J} \\
\vdots \ & \vdots \ & \ddots \ & \vdots \\
J, \beta_{J,1}, \beta_{J,2}, \cdots, \beta_{J,J} \\
\end{bmatrix}
\]
(57)
with the sizes \(\ell \times 1, \ell \times (n+1)J\), and \((n+1)J \times 1\), respectively. Again, the iterative divide-and-merge-based least squares estimator is adopted to find an optimal \(\beta\) in (57).

VII. EXPERIMENTAL RESULTS

We demonstrate the performance of our approach by showing the results of several experiments. In Experiment I, we use a simple function to illustrate how our approach works. In the subsequent two experiments, we run on another function and a real-world dataset, respectively. A comparison with two other methods, adopting type-1 fuzzy sets [5] and interval type-2 fuzzy sets [52], respectively, is presented. For convenience, in the following our approach is abbreviated as T2FNN and the other two as T1FNN and IT2FNN, respectively. We use a computer with Intel(R) Xeon(TM) CPU X5420 2.50 GHz and 4 GB of RAM to conduct the experiments. The operating system is CentOS 5.3. The programming language used is JAVA 1.6.

In T1FNN, the fuzzy sets involved in the antecedent part of a fuzzy rule are type-1 fuzzy sets and have Gaussian membership functions. That is, all the \(\tilde{A}_{i,j}, 1 \leq i \leq n, 1 \leq j \leq J\), in (16) are type-1 fuzzy sets having the membership function in the form as shown in (7). In IT2FNN, the fuzzy sets involved in the antecedent part of a fuzzy rule are interval type-2 fuzzy sets. The principal membership function of each interval type-2 fuzzy set is bound by two Gaussian functions with distinct means. The consequent part in both T1FNN and IT2FNN has the same form as that shown in (16). For T1FNN, the antecedent parameters to be refined include the mean and the deviation of each type-1 fuzzy set. For IT2FNN, the antecedent parameters to be refined include two means and two deviations associated with each interval type-2 fuzzy set. The consequent parameters to be refined for both T1FNN and IT2FNN are totally the same as for T2FNN.

A. Experiment I

The first experiment concerns the modeling of the following nonlinear function
\[
y = x_1^2 \times \sin(x_2 \pi)
\]
where \(x_1\) and \(x_2\) are the inputs, \(x_1, x_2 \in [-1, 1]\), and \(y\) is the output. The function is shown graphically in Fig. 6(a).

We take 15 samples randomly in \(x_1\) and \(x_2\), respectively, and have \(15 \times 15 = 225\) training patterns, i.e., \(\ell = 225\).

Step 1. Structure identification phase.

We run the self-constructing fuzzy clustering algorithm on the training patterns with \(\rho = 0.0001\) and \(\tau = 0.4\), and \(\sigma_0 = 0.1\). Six clusters \(G_1, G_2, G_3, G_4, G_5, G_6\) are obtained as shown in Table I.

According to these clusters, the following initial type-2 fuzzy rules are developed.

1) \(R_1:\) IF \(x_1\) is \(\text{gauss}(u; \text{gauss}(x_1; -0.7009, 0.4523), 0.04523)\)
AND \(x_2\) is \(\text{gauss}(u; \text{gauss}(x_2; -0.6518, 0.4762), 0.04762)\)
THEN \(y\) is \(c_1 = -0.1796 + 0.6368 x_1 - 0.4315 x_2\).

2) \(R_2:\) IF \(x_1\) is \(\text{gauss}(u; \text{gauss}(x_1; -0.0849, 0.6968), 0.06968)\)
AND \(x_2\) is \(\text{gauss}(u; \text{gauss}(x_2; 0.2728, 0.7179), 0.07179)\)
THEN \(y\) is \(c_2 = 0.0508 - 0.0792 x_1 + 0.1204 x_2\).

3) \(R_3:\) IF \(x_1\) is \(\text{gauss}(u; \text{gauss}(x_1; 0.3786, 0.3486), 0.03486)\)
AND \(x_2\) is \(\text{gauss}(u; \text{gauss}(x_2; -0.6143, 0.4275), 0.04275)\)
THEN \(y\) is \(c_3 = -0.0090 - 0.5690 x_1 - 0.1706 x_2\).

4) \(R_4:\) IF \(x_1\) is \(\text{gauss}(u; \text{gauss}(x_1; 0.8435, 0.3348), 0.03348)\)
AND \(x_2\) is \(\text{gauss}(u; \text{gauss}(x_2; -0.6122, 0.4898), 0.04898)\)
THEN \(y\) is \(c_4 = 0.1248 - 0.1475 x_1 - 0.4992 x_2\).

5) \(R_5:\) IF \(x_1\) is \(\text{gauss}(u; \text{gauss}(x_1; 0.9184, 0.2923), 0.02923)\)
AND \(x_2\) is \(\text{gauss}(u; \text{gauss}(x_2; 0.4694, 0.5040), 0.05040)\)
THEN \(y\) is \(c_5 = -0.5280 + 1.0188 x_1 + 0.2667 x_2\).
of 0.0259 with respect to the training patterns. For example, when \( x_1 = 0.57143 \) and \( x_2 = -0.71429 \), the desired \( y \) value is \(-0.25529\) by (60). Using the six rules, a fuzzy network in the form of Fig. 3 is built and we can approximate the \( y \) value as follows. Layer 1 computes the values of the corresponding membership functions. For example, \( o^{(1)}_{1,1} \) and \( o^{(1)}_{2,1} \) have the following values:

\[
\begin{align*}
o^{(1)}_{1,1} &= \text{gauss}(u; \text{gauss}(0.57143, -0.7009, 0.4523, 0.04523)) \\
o^{(1)}_{2,1} &= \text{gauss}(u; \text{gauss}(-0.71429, -0.6518, 0.4762, 0.04762)).
\end{align*}
\]

Layer 2 computes the firing strength of each rule. For example, the firing strength of \( R_1 \) is

\[
o^{(2)}_1 = o^{(1)}_{1,1} \times o^{(1)}_{2,1}
\]

which is a type-1 fuzzy set. Layer 3 performs type reduction on its input, which is a type-2 fuzzy set, and produces output \( o^{(3)} \) which is a type-1 fuzzy set. Finally, layer 4 performs defuzzification on \( o^{(3)} \) and its output \( o^{(4)} \) is \(-0.16285\). Therefore, the inferred output for \( x_1 = 0.57143 \) and \( x_2 = -0.71429 \) from the above six rules is \( \hat{y} = o^{(4)} = -0.16285 \). The square error induced is \((y - \hat{y})^2 = 0.00855\).

**Step 2. Parameter identification phase.**

We improve the precision of the rules through the application of the hybrid learning algorithm. For PSO, the population size is set as 10, and the parameters \( w, k_1, \) and \( k_2 \) are set as 0.5, 1.0, and 1.0, respectively. The refined type-2 fuzzy rules obtained are as follows.

1) \( R_1 \): IF \( x_1 \) is \( \text{gauss}(u; \text{gauss}(x_1; -0.2133, 1.2042), 0.0001) \) AND \( x_2 \) is \( \text{gauss}(u; \text{gauss}(x_2; 0.5924, 1.1628), 0.0118) \) THEN \( y \) is \( c_1 = 0.0722 + 0.2398x_1 + 0.0337x_2 \).

2) \( R_2 \): IF \( x_1 \) is \( \text{gauss}(u; \text{gauss}(x_1; -1.0801, 0.4464), 0.0022) \) AND \( x_2 \) is \( \text{gauss}(u; \text{gauss}(x_2; -0.7499, 0.4334), 0.0013) \) THEN \( y \) is \( c_2 = -2.0984 + 0.6210x_1 - 2.5029x_2 \).

3) \( R_3 \): IF \( x_1 \) is \( \text{gauss}(u; \text{gauss}(x_1; 0.7509, 0.4421), 0.0013) \) AND \( x_2 \) is \( \text{gauss}(u; \text{gauss}(x_2; -0.1818, 0.4888), 0.0007) \) THEN \( y \) is \( c_3 = 2.4349 - 4.9364x_1 + 1.2439x_2 \).

4) \( R_4 \): IF \( x_1 \) is \( \text{gauss}(u; \text{gauss}(x_1; 0.3829, 0.4691), 0.1992) \) AND \( x_2 \) is \( \text{gauss}(u; \text{gauss}(x_2; -0.6381, 0.8518), 0.1709) \) THEN \( y \) is \( c_4 = -0.7471 + 0.1108x_1 - 0.7452x_2 \).

\[
\begin{array}{|c|c|c|c|}
\hline
\text{Rule} & \text{Mean } m_j & \text{Standard deviation } \sigma_j & \text{Height } h_j \\
\hline
G_1 & [-0.7009, -0.6518] & [0.4523, 0.4762] & -0.3447 \\
G_2 & [-0.0849, 0.2728] & [0.6968, 0.7179] & 0.0904 \\
G_3 & [0.3786, -0.6143] & [0.3486, 0.4275] & -0.1196 \\
G_4 & [0.8435, -0.6122] & [0.3348, 0.4898] & -0.4532 \\
G_5 & [0.9184, 0.4694] & [0.2923, 0.5040] & 0.5329 \\
G_6 & [-0.8857, 0.6286] & [0.3195, 0.3629] & 0.6924 \\
\hline
\end{array}
\]

Table I

**Six Clusters Obtained for Experiment I**

Fig. 6. Experiment I. (a) Output of the desired function. (b) Output of the six initial rules. (c) Output of the six refined rules.
5) \( R_5: \text{IF } x_1 \text{ is } \text{gauss}(u; \text{gauss}(x_1; 0.9167, 0.3129), 0.2120) \)
\( \text{AND } x_2 \text{ is } \text{gauss}(u; \text{gauss}(x_2; 0.7078, 0.8724), 0.0004) \)
\( \text{THEN } y \text{ is } c_5 = 2.3696 + 0.9077x_1 - 3.2286x_2. \)

6) \( R_6: \text{IF } x_1 \text{ is } \text{gauss}(u; \text{gauss}(x_1; -0.8344, 0.4774), 0.0008) \)
\( \text{AND } x_2 \text{ is } \text{gauss}(u; \text{gauss}(x_2; 0.5073, 0.3346), 0.2629) \)
\( \text{THEN } y \text{ is } c_6 = -0.4988 - 2.3208x_1 + 0.1153x_2. \)

The output of these refined rules is shown in Fig. 6(c) with an MSE of 0.0012 with respect to the training patterns. Apparently, the refined rules give a better approximation to the original function than the initial rules. For example, when \( x_1 = -0.5714 \) and \( x_2 = -0.7143 \), the inferred output from the refined rules is \( \hat{y} = -0.25472 \). The square error induced is \( (y - \hat{y})^2 = 3.25 \times 10^{-7} \).

### B. Experiment II

This experiment investigates the noise resistance of each method. Consider the following nonlinear function [5], [47]:

\[
y = \sin(x_1 \pi) \cdot \sin(x_2 \pi)
\]

where \( x_1 \) and \( x_2 \) are the inputs, \( y \) is the output, and \( x_1 \in [-1, 1] \) and \( x_2 \in [0, 1] \). The input–output training data pairs, \( \langle [p_1, p_2], d \rangle \), are taken by sampling \( x_1 \) and \( x_2 \) with the sampling interval being 0.1. As a result, there are totally 231 input–output training patterns, i.e., \( \ell = 231 \). We also take 200 samplings in random for testing. The values of \( \rho, \tau, \) and \( \sigma_0 \) are set to be 0.01^2, 0.2, and 0.15, respectively, for the self-constructing fuzzy clustering algorithm. We consider three versions of the training dataset. Version 1 contains uncorrupted data, i.e., no noise added. In Version 2, the training data are corrupted by adding a noise which is a normal distribution with zero mean and standard deviation being 0.2. In Version 3, the noise added is a normal distribution with zero mean and standard deviation being 0.3. Note that noise is added to all of the patterns in Version 2 and Version 3.

Table II shows the results for the Version 1 training dataset. Note that 6 rules are obtained. The column “Training MSE” indicates the induced error with respect to the training patterns, while the column “Testing MSE” indicates the induced error with respect to the testing patterns. The column “Time” indicates the CPU time, in seconds, spent in modeling, i.e., the time spent in structure and parameter identification phases. From this table, we can see T2FNN achieves the best training and testing MSE. For example, T2FNN gets 0.0056 in training MSE and 0.00040 in testing MSE, while T1FNN gets 0.00173 and 0.00111 and IT2FNN gets 0.00095 and 0.00066, respectively. However, T1FNN runs faster, taking 1.806 s in this case. T2FNN and IT2FNN run comparably slow. Note that computation time is not an important issue when systems are not going to be run in real time. Table III shows the results for the Version 2 training dataset. Apparently, T2FNN achieves the best training and testing MSE. For example, T2FNN gets 0.00396 in training MSE and 0.00051 in testing MSE, while T1FNN gets 0.04762 and 0.01085 and IT2FNN get 0.04185 and 0.00781, respectively. Table IV shows the results for the Version 3 training dataset. Obviously, T2FNN achieves the best training and testing MSE for this case. For example, T2FNN get 0.07856 in training MSE and 0.00997 in testing MSE, while T1FNN gets 0.08074 and 0.01943, and IT2FNN gets 0.08422 and 0.01626, respectively. Note that IT2FNN has a worse training error than T1FNN for this case. T1FNN runs faster, taking 2.672 s in this case, and T2FNN and IT2FNN run slower.

For PSO, the population size is set as 10. Changing the size does not make much difference in the results. Table V shows the results obtained with the population size being 20 for the three versions of the training data. The testing MSEs are only slightly better than those shown in Tables II–IV. For example, for the Version 1 training dataset the testing MSE is 0.00040 with size being 10 and is 0.00039 with size being 20. The difference between them is small, within 3%. This indicates that the initial fuzzy rules obtained from the clustering algorithm already provide a pretty good approximation. Therefore, a small population size, e.g., 10, in PSO is sufficient in parameter identification.

A standard neural network provided in MATLAB [61] is also tried for the above three cases. The number of hidden nodes is set to be 6, gradient descent is applied for training with 0.1 as the learning rate, and 1000 epochs are done before.
termination. For each case, we run 200 times and choose the best performance for comparison. The testing MSE for the version of uncorrupted training data is 0.0028. For the other two versions, they are 0.0236 and 0.0296, respectively.

### C. Experiment III

The last experiment concerns a real-world dataset, miles per gallon (MPG), taken from UCI Repository of Machine Learning Databases [62]. The MPG dataset is used for evaluating the city-cycle fuel consumption in miles per gallon. This dataset contains 392 patterns, each pattern consisting of seven input values and one output value. Tenfold cross-validation is applied, i.e., nine-tenths of the patterns are used for training and the remaining one-tenth for testing. Ten runs were conducted and the outcomes of these 10 runs are averaged. Table VI shows the results with \( \tau = 0.5 \), \( \sigma_0 = 0.2 \), and three different \( \rho \) values. Note that \( \rho \) is a user-defined threshold for input similarity test in the self-constructing clustering algorithm. The setting of \( \rho \) may affect the number of clusters obtained. As \( \rho \) increases, the patterns in a cluster are required to be more similar to each other and thus the number of clusters obtained also increases. The column “MSE” in Table VI indicates the induced error with respect to the testing patterns. As seen from the table, T2FNN achieves the best MSE for the testing data. For example, T2FNN gets 0.0070 in MSE, while T1FNN and IT2FNN get 0.0091 and 0.0088, respectively, with 2.7 rules. However, T2FNN runs slower than the other two methods. For example, T1FNN and IT2FNN require 3.037 and 4.071 s, respectively, in modeling, while T2FNN requires 5.290 s in this case. To rule out random effects in comparisons, paired samples \( t \)-tests were conducted for two pairs of competitors, T2FNN versus T1FNN and T2FNN versus IT2FNN. A paired sample \( t \)-test [63] can be used to compare the means of two variables. It computes the difference between the two variables and tests to see if the average difference is significantly different from zero. In this experiment, we compare different methods by using the same sets of training and testing data each time. By regarding each method as a variable, we can apply a paired sample \( t \)-test to test if there is a significant difference between the means obtained from the involved two methods. Since we run 10-fold cross validation, 10 pieces of data are collected for each variable in a paired sample \( t \)-test. The \( p \)-values of the tests are shown in Table VII. Note that the lower the \( p \)-value, the more significant the result in the sense of statistical significance. In this case, if the \( p \)-value is less than 0.05, then there is a significant difference between the two competitors. Apparently, the \( p \)-values show that T2FNN is significantly better than T1FNN and IT2FNN in terms of MSE.

Table VIII shows the results with \( \rho = 0.1^1 \), \( \sigma_0 = 0.2 \), and three different \( \tau \) values. Note that \( \tau \) is a user-defined threshold for output similarity test in the self-constructing clustering algorithm. The setting of \( \tau \) may affect the number of clusters obtained. As \( \tau \) decreases, the outputs of the patterns in a cluster are required to be closer to each other and thus the number of clusters obtained increases. Apparently, T2FNN achieves the best MSE for the testing data. For example, T2FNN gets 0.0079 in MSE, while T1FNN and IT2FNN get 0.0090 and 0.0097, respectively, with 3.0 rules. However, T2FNN runs slower than the other two methods. For example, T1FNN and IT2FNN require 3.792 and 4.707 s, respectively, in modeling, while T2FNN requires 6.852 s in this case. The \( p \)-values of paired samples \( t \)-tests for the two pairs, T2FNN versus T1FNN and T2FNN versus IT2FNN, are shown in Table IX. Obviously, the \( p \)-values show that T2FNN is significantly better than T1FNN and IT2FNN in terms of MSE.

Table X shows the results with \( \rho = 0.1^1 \), \( \tau = 0.5 \), and three different \( \sigma_0 \) values. Note that \( \sigma_0 \) is the initial value for the deviation in the self-constructing clustering algorithm. The setting of \( \sigma_0 \) may affect the number of clusters. As \( \sigma_0 \)
decreases, the patterns in a cluster are required to be more similar to each other and thus the number of clusters obtained increases. Again, T2FNN achieves the best MSE for the testing data. For example, T2FNN gets 0.0076 in MSE, while T1FNN and IT2FNN get 0.0090 and 0.0082, respectively, with 2.4 rules. However, T2FNN runs slower than the other two methods. For example, T1FNN and IT2FNN require 2.699 and 3.730 s, respectively, in modeling, while T2FNN requires 5.091 s in this case. The p-values of paired samples t-tests for the two pairs, T2FNN versus T1FNN and T2FNN versus IT2FNN, are shown in Table XI. Again, the p-values show that T2FNN is significantly better than T1FNN and IT2FNN in terms of MSE.

VIII. CONCLUSION

We have presented a novel approach for developing type-2 fuzzy rules from a given set of input–output training data. A self-construction fuzzy clustering method is used to partition the training dataset into clusters through input-similarity and output-similarity tests. The membership functions of the resulting clusters are defined with statistical means and deviations. One type-2 fuzzy TSK-rule is derived from each cluster. A fuzzy neural network is constructed accordingly and the associated parameters are refined by a hybrid learning algorithm which incorporates PSO and LSE. For a new input, a corresponding crisp output of the system is obtained by combining the inferred results of all the rules into a type-2 fuzzy set which is then defuzzified by applying a refined type reduction algorithm. The effectiveness of the proposed approach was demonstrated through several experiments.

Fuzzy clustering and type reduction are two topics we have been working on in our machine learning research. In this paper, we applied them to building a type-2 neural-fuzzy system from a given set of input–output training data. We are also applying them to other problems, such as image segmentation, data sampling, feature reduction, web mining, etc. One restriction of our type reduction algorithm is that the secondary membership functions of a type-2 fuzzy set are required to be convex. A type-1 fuzzy set is convex if its every α-cut, \( \alpha > 0 \), is a continuous interval. We are working on efficient type reduction algorithms for non-convex cases. Besides, we are seeking more applications of type-2 fuzzy systems.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their comments, who were very helpful in improving the quality and presentation of this paper.

TABLE XI

<table>
<thead>
<tr>
<th>( \sigma_0 )</th>
<th>T2FNN versus T1FNN</th>
<th>T2FNN versus IT2FNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>0.00001</td>
<td>0.0152</td>
</tr>
<tr>
<td>0.20</td>
<td>0.0019</td>
<td>0.0032</td>
</tr>
<tr>
<td>0.25</td>
<td>0.0001</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

REFERENCES

Shie-Jue Lee (M’88) was born in Kin-Men on August 15, 1955. He received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1977 and 1979, respectively, and the Ph.D. degree in computer science from the University of North Carolina, Chapel Hill, in 1990.

He joined the faculty of the Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, in 1983, and has been a Professor in the department since 1994. His current research interests include artificial intelligence, machine learning, data mining, information retrieval, and soft computing.

Prof. Lee is a member of the IEEE Society of Systems, Man, and Cybernetics, the Association for Automated Reasoning, the Institute of Information and Computing Machinery, the Taiwan Fuzzy Systems Association, and the Taiwanese Association of Artificial Intelligence. He was the recipient of the Outstanding M.S. Thesis Supervision Award by the Chinese Institute of Electrical Engineering for 1997, and also the Best Paper Award in several international conferences. Among the other awards won by him are the Distinguished Teachers Award of the Ministry of Education, Taiwan, in 1993, the Distinguished Research Award of National Sun Yat-Sen University in 1998, the Distinguished Teaching Award of National Sun Yat-Sen University in 1993 and 2008, and the Distinguished Mentor Award of National Sun Yat-Sen University in 2008. He served as the Program Chair for the International Conference on Artificial Intelligence, Kaohsiung, Taiwan, in December 1996, the International Computer Symposium and Workshop on Artificial Intelligence, Tainan, Taiwan, in December 1998, and the 6th Conference on Artificial Intelligence and Applications, Kaohsiung, in November 2001. He served as the Director of the Center for Telecommunications Research and Development from 1997 to 2000, the Director of the Southern Telecommunications Research Center, National Science Council, from 1998 to 1999, the Chair of the Department of Electrical Engineering from 2000 to 2003, and the Deputy Dean of the Academic Affairs from 2008 to 2011. He is now the Director of the Library and Information Services and the Director of the NSYSU-III Research Center, National Sun Yat-Sen University.