An ART-Based Fuzzy Adaptive Learning Control Network

Cheng-Jian Lin, *Member, IEEE*, and Chin-Teng Lin, *Member, IEEE*

***Abstract*—This paper addresses the structure and an associated on-line learning algorithm of a feedforward multilayered connectionist network for realizing the basic elements and functions of a traditional fuzzy logic controller. The proposed fuzzy adaptive learning control network (FALCON) can be contrasted with the traditional fuzzy logic control systems in their network structure and learning ability. An on-line structure/parameter learning algorithm called FALCON-ART is proposed for constructing the FALCON dynamically. It combines the backpropagation learning scheme for parameter learning and the fuzzy ART algorithm for structure learning. The FALCON-ART has some important features. First of all, it partitions the input state space and output control space using irregular fuzzy hyperboxes according to the distribution of training data. In many existing fuzzy or neural fuzzy control systems, the input and output spaces are always partitioned into "grids." As the number of input/output variables increases, the number of partitioned grids will grow combinatorially. To avoid the problem of combinatorial growing of partitioned grids in some complex systems, the FALCON-ART partitions the input/output spaces in a flexible way based on the distribution of training data. Second, the FALCON-ART can create and train the FALCON in a highly autonomous way. In its initial form, there is no membership function, fuzzy partition, and fuzzy logic rule. They are created and begin to grow as the first training pattern arrives. Thus, the users need not give it any *a priori* knowledge or even any initial information on these. More notably, the FALCON-ART can on-line partition the input/output spaces, tune membership functions, find proper fuzzy logic rules, and annihilate redundant rules dynamically upon receiving on-line incoming training data. Computer simulations have been conducted to illustrate the performance and applicability of the proposed system.**

***Index Terms*—Adaptive vector quantization, fuzzy ART, fuzzy clustering, fuzzy hyperbox, rule annihilation, time-series prediction.**

I. Introduction

BRINGING the learning abilities of neural networks to automate and realize the design of fuzzy logic control systems has recently become a very active research area [1]–[18]. This integration brings the low-level learning and computation power of neural networks into fuzzy logic systems and provides the high-level human-like thinking and reasoning of fuzzy logic systems into neural networks. Such synergism of integrating neural networks and fuzzy logic into a functional system provides a new direction toward the realization of intelligent systems for various applications.

In this paper, we are extending our previous work on neural network-based fuzzy logic control systems [19] to the on-line supervised learning problems. The proposed fuzzy adaptive learning control network (FALCON) can be constructed automatically by learning from training examples. It can be contrasted with the traditional fuzzy logic control systems in their network structure and learning ability. The FALCON is a five-layer structure, as shown in Fig. 1. Nodes at layer one are input nodes (linguistic nodes), which represent input linguistic variables. Layer five is the output layer. We have two linguistic nodes for each output variable. One is for training data (desired output) to feed into this net and the other is for decision signal (actual output) to be pumped out of the net. Nodes at layers two and four are term nodes, which act as membership functions to represent the terms of the respective linguistic variable. Each node at layer three is a rule node which represents one fuzzy logic rule. Thus, all layer-three nodes form a fuzzy rule base. Layer-three links define the preconditions of the rule nodes and layer-four links define the consequents of the rule nodes. The links at layers two and five are fully connected between linguistic nodes and their corresponding term nodes.

Associated with the FALCON is an on-line learning algorithm called FALCON-ART. We shall call a FALCON with this on-line learning algorithm the FALCON-ART model. The FALCON-ART has some important properties, as described below. In many existing fuzzy or neural fuzzy control systems, the input and output spaces are partitioned into "grids." Each grid defines a fuzzy region and the overlapping region between the grids provides a smooth and continuous membership output surface. For example, consider a fuzzy logic controller with two input variables. If each of them contains three fuzzy terms (e.g., "small," "medium," and "large"), then the corresponding input space partition is as shown in Fig. 2(a). Although during the learning process, the position and shape of membership functions will be changed, they are still grid-type partitions inherently. The grid-type space partitioning of input and output spaces has been widely used in many existing fuzzy systems. It certainly makes both the fuzzy logic controller software emulation and fuzzy chip implementation convenient. However, as the number of input/output variables increases, the number of the partitioned grids will grow combinatorially. As a result, the required size of memory or hardware may become impractically huge. This results in more difficulty in learning because finer space partitioning needs more training samples; otherwise, insufficient learning will occur.

Manuscript received March 8, 1994; revised December 12, 1996. This work was supported by the R.O.C. National Science Council under Grant NSC86-2212-E-009-019.

The authors are with the Department of Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C.

Publisher Item Identifier S 1063-6706(97)04832-7.

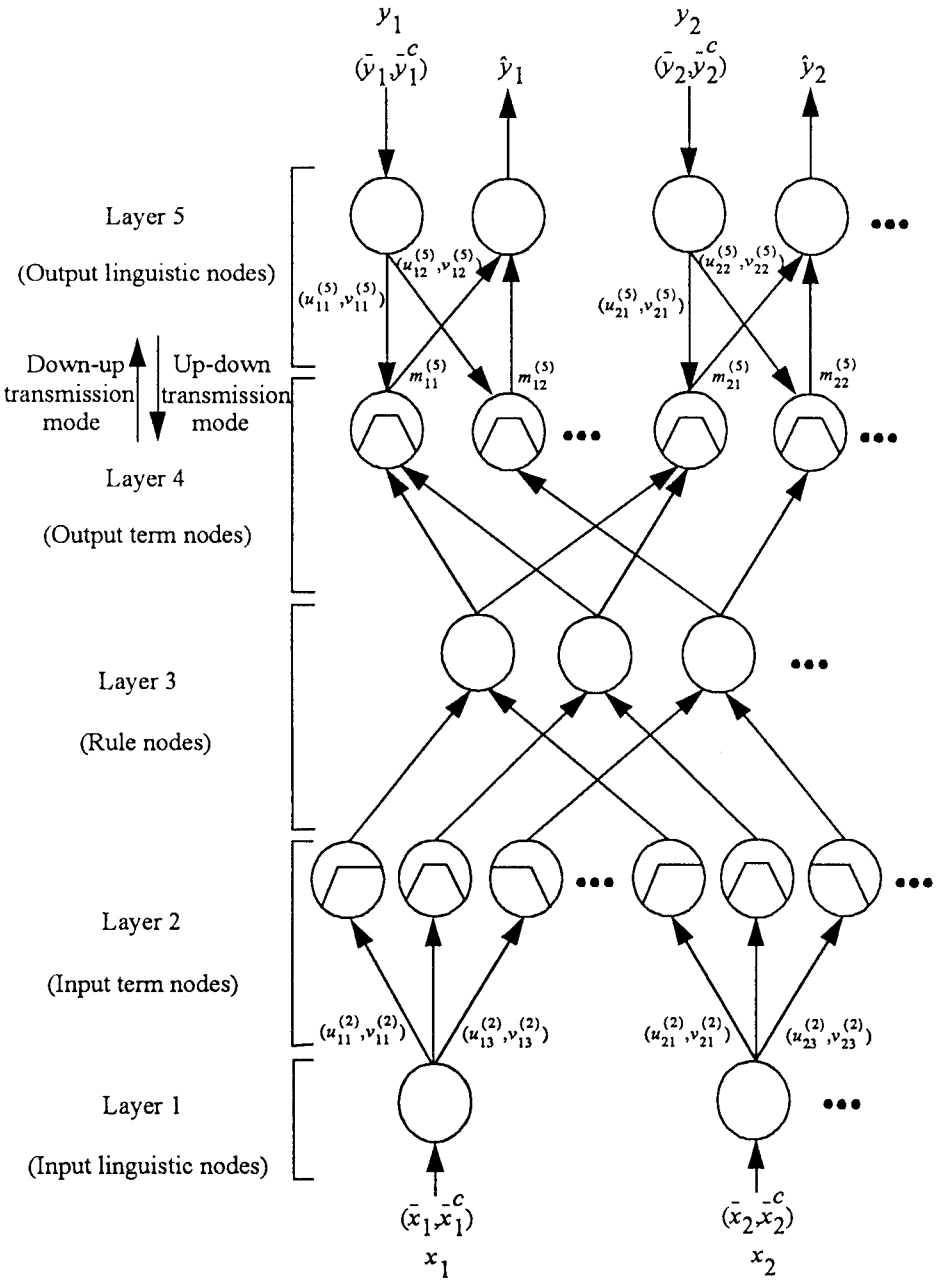
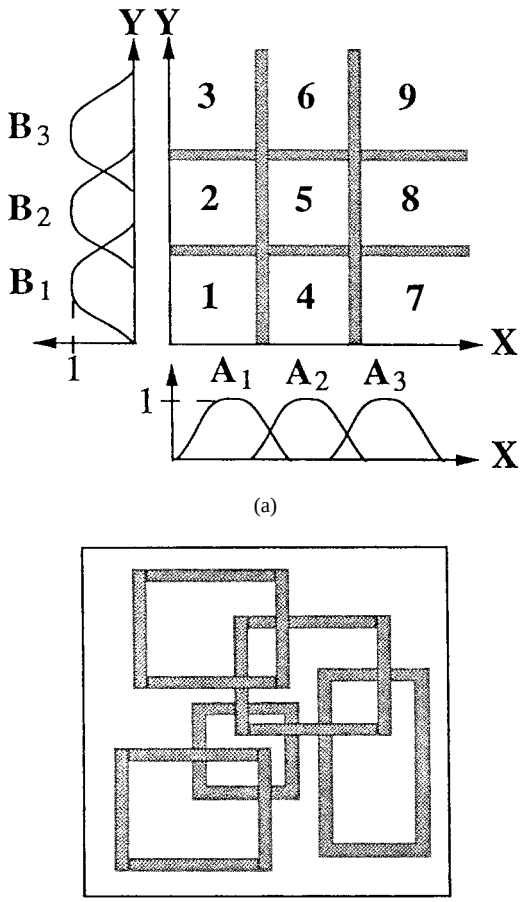


Fig. 1. Proposed FALCON.

To avoid the problem of combinatorial growing of partitioned grids in complex systems, more flexible and irregular space partitioning methods must be developed. Fig. 2(b) shows a proposed partitioning method in the FALCON-ART. The problem of space partitioning from numerical training data is basically a clustering problem. The proposed FALCON-ART applies the fuzzy adaptive resonance theory (fuzzy ART) proposed by Carpenter *et al.* [22], [23] to do fuzzy clustering in the input/output spaces and find proper fuzzy logic rules dynamically by associating input clusters and output clusters. The backpropagation learning scheme is then used for tuning input/output membership functions. Hence, the FALCON-ART combines the backpropagation algorithm for parameter learning and the fuzzy ART for structure learning. The FALCON-ART can, thus, on-line partition the input/output spaces, tune membership functions, and find proper fuzzy logic rules dynamically in the fly. More notably, in this learning method, only the training data need to be provided from the outside world. The users need not give the initial fuzzy partitions, membership functions, and fuzzy logic rules. Hence, there is no input/output term nodes and no rule nodes in the beginning of learning. They are created dynamically as learning proceeds upon receiving on-line incoming training data.

This paper is organized as follows. Section II describes the structure of the FALCON-ART model. The on-line structure/parameter learning algorithm FALCON-ART, which combines fuzzy ART with backpropagation, is presented



(a)

(b)

Fig. 2. (a) Grid-type fuzzy partitioning. (b) Flexible hyperbox fuzzy partitioning.

in Section III. The structure learning consists of three learning processes: input fuzzy clustering process, output fuzzy clustering process, and mapping process. Section IV describes the proposed rule annihilation methods used in the learning process. In Section V, the FALCON-ART model is used to identify the nonlinear dynamic system, predict the Mackey–Glass chaotic time-series, control the truck backer-upper, and control the ball and beam system to demonstrate its on-line learning capability. Section VI describes the features of the proposed FALCON-ART model. Conclusions are summarized in the last section.

II. The Structure of the FALCON-ART Model

In this section, we shall describe the structure and functions of the proposed FALCON-ART model. The FALCON (see Fig. 1) has five layers with node and link numbering defined by the brackets on the right-hand side of the figure. Layer-1 nodes are input nodes (*input-linguistic nodes*) representing input-linguistic variables. Layer-5 nodes are output nodes (*output-linguistic nodes*) representing output linguistic variables. Layer-2 and Layer-4 nodes are *term nodes* that act as membership functions representing the terms of the respective input- and output-linguistic variables. Each Layer-3 node is a rule node representing one fuzzy logic rule. Thus, together, all the Layer-3 nodes form a fuzzy rule base. Links between Layers 3 and 4 function as a *connectionist inference engine*. Layer-3 links define the preconditions of the rule nodes and Layer-4 links define the consequents of the rule nodes. Therefore, each rule node has at most one link to some term node of a linguistic node, and may have no such links. This is true both for precondition links (links in Layer 3) and consequent links (links in Layer 4). The links in Layers 2 and 5 are fully connected between linguistic nodes and their corresponding term nodes. The arrows indicate the normal signal flow directions when the network is in operation (after it has been built and trained). We shall later indicate the signal propagation layer-by-layer, according to the arrow direction.

The FALCON uses the technique of *complement coding* from fuzzy ART [22] to normalize the input/output training vectors. Complement coding is a normalization process that rescales an n-dimensional vector in $\Re^n$, $\mathbf{x} = (x_1, x_2, \cdots, x_n)$, to its $2n$-dimensional complement coding form in $[0, 1]^{2n}$, $\mathbf{x}'$, such that

$$\begin{aligned}\mathbf{x}' &\equiv (\overline{x}_1, \overline{x}_1^c, \overline{x}_2, \overline{x}_2^c, \cdots, \overline{x}_n, \overline{x}_n^c) \\ &= (\overline{x}_1, 1-\overline{x}_1, \overline{x}_2, 1-\overline{x}_2, \cdots, \overline{x}_n, 1-\overline{x}_n) \end{aligned} \quad (1)$$

where $(\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_n) = \overline{\mathbf{x}} = \mathbf{x}/\|\mathbf{x}\|$, and $\overline{x}_1^c$ is the complement of $\overline{x}_1$, i.e., $\overline{x}_1^c = 1 - \overline{x}_1$. As mentioned in [22], complement coding helps avoid the problem of category proliferation when using fuzzy ART for fuzzy clustering. It also preserves training vector-amplitude information. In applying the complement coding technique to the FALCON, all training vectors (either input state vectors or desired output vectors) are transformed to their complement coded forms in the preprocessing process and the transformed vectors are then used for training.

A typical neural network consists of nodes with some finite number of fan-in connections from other nodes represented by weight values and fan-out connections to other nodes. Associated with the fan-in of a node is an integration function f which combines information, activation, or evidence from other nodes, and provides the net input, i.e.,

$$\begin{aligned}&\text{net-input} \\ &\quad = f[z_1^{(k)}, z_2^{(k)}, \cdots, z_p^{(k)}; w_1^{(k)}, w_2^{(k)}, \cdots, w_p^{(k)}] \end{aligned} \quad (2)$$

where $z_i^{(k)}$ is the ith input to a node in layer k and $w_i^{(k)}$ is the weight of the associated link. The superscript in the above equation indicates the layer number. This notation will also be used in the following equations. Each node also outputs an activation value as a function of its net-input

$$\text{output} = a[f(\cdot)] \quad (3)$$

where $a(\cdot)$ denotes the activation function. Next, we shall describe the functions of the nodes in each of the five layers of the FALCON. Assume that the dimension of the input space is n and that of the output space is m.

Layer 1: Each node in this layer is called an input-linguistic node and corresponds to one input-linguistic variable. Layer-1 nodes just transmit input signals to the next layer directly. That is

$$f(\overline{x}_i, \overline{x}_i^c) = (\overline{x}_i, \overline{x}_i^c) = (\overline{x}_i, 1 - \overline{x}_i)$$

and

$$a[f(\cdot)] = f(\cdot). \tag{4}$$

From the above equation, the link weight in Layer 1 $[w_i^{(1)}]$ is unity. Notice that due to the complement coding process, for each input node i there are two output values $\overline{x}_i$ and $\overline{x}_i^c = 1 - \overline{x}_i$.

Layer 2: Nodes in this layer are called input-term nodes and each represents a term of an input-linguistic variable and acts as a one-dimensional (1-D) membership function. The following trapezoidal membership function [24] is used:

$$f[z_{ij}^{(2)}] = \{1 - g[z_{ij}^{(2)} - v_{ij}^{(2)}, \gamma] - g[u_{ij}^{(2)} - z_{ij}^{(2)}, \gamma]\}$$

and

$$a[f(\cdot)] = f(\cdot) \tag{5}$$

where $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ are, respectively, the left-flat and right-flat points of the trapezoidal membership function of the jth input-term node of the ith input-linguistic node [see Fig. 3(a)], $z_{ij}^{(2)}$ is the input to the jth input-term node from the ith input-linguistic node (i.e., $z_{ij}^{(2)} = \overline{x}_i$), and

$$g(s, \gamma) = \begin{cases} 1, & \text{if } s\gamma > 1, \\ s\gamma, & \text{if } 0 \le s\gamma \le 1, \\ 0, & \text{if } s\gamma < 0. \end{cases} \tag{6}$$

The parameter γ is the sensitivity parameter that regulates the fuzziness of the trapezoidal membership function. A large γ means a more crisp fuzzy set, and a smaller γ makes the fuzzy set less crisp. A set of n input-term nodes (one for each input-linguistic node) is connected to a rule node in Layer 3 where its outputs are combined. This defines an n-dimensional membership function in the input space with each dimension specified by one input-term node in the set. Hence, each input-linguistic node has the same number of term nodes. That is, each input-linguistic variable has the same number of terms in the FALCON. This is also true for output-linguistic nodes. A Layer-2 link connects an input-linguistic node to one of its term nodes. There are two weights on each Layer-2 link. We denote the two weights on the link from input node i (corresponding to the input-linguistic variable x_i) to its jth term node as $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ (see Fig 1). These two weights define the membership function in (5). The two weights $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ correspond, respectively, to the two inputs $\overline{x}_i$ and $\overline{x}_i^c$ from the input-linguistic node i. More precisely, $\overline{x}_i$ and $\overline{x}_i^c$, the two inputs to the input-term node j, will be used during the fuzzy-ART clustering process in FALCON's structure-learning step to decide $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, respectively. In FALCON's parameter-learning step and normal operating, only $\overline{x}_i$ is used in the forward reasoning process [i.e., $z_{ij}^{(2)} = \overline{x}_i$ in (5)]. We detail the FALCON learning scheme in Section III.

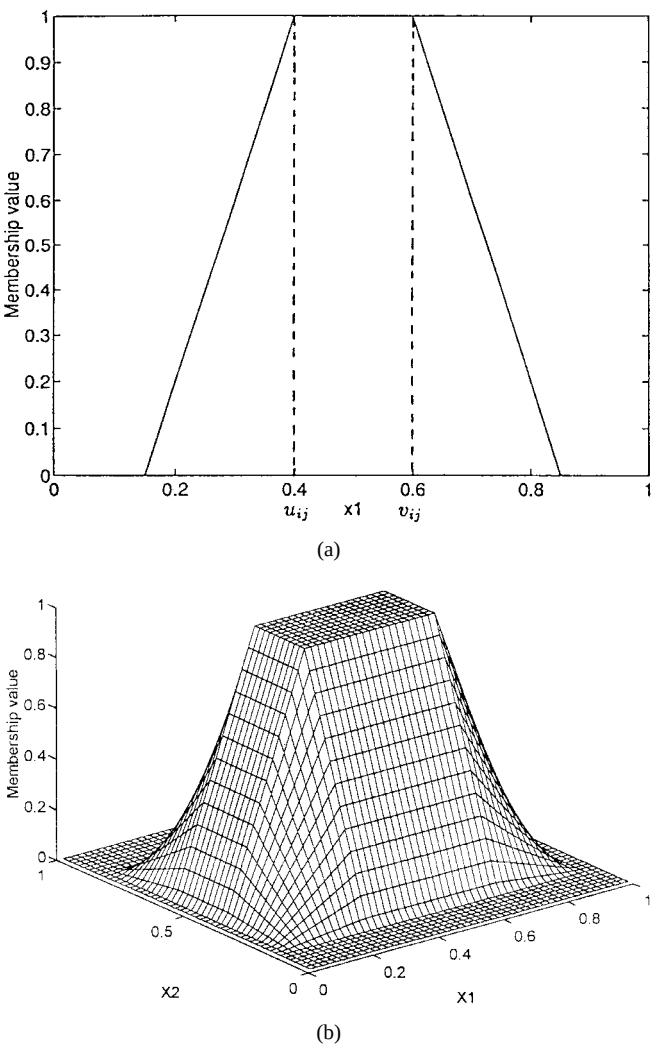


Fig. 3. (a) One-dimensional (1-D) trapezoidal membership function. (b) Two-dimensional (2-D) trapezoidal membership function.

Layer 3: Nodes in this layer are called rule nodes and each represents one fuzzy logic rule. Each Layer-3 node has n input-term nodes fed into it, one for each input-linguistic node. Hence, there are as many rule nodes in the FALCON as there are term nodes of an input-linguistic node (i.e., the number of rules equals the number of terms of an input-linguistic variable). Notice that each input-linguistic variable has the same number of terms in the FALCON, as mentioned in the above. The links in Layer 3 are used to perform precondition matching of fuzzy logic rules. Hence, the rule nodes perform the product operation

$$f[z_i^{(3)}] = \prod_{i=1}^{n} z_i^{(3)}$$

and

$$a[f(\cdot)] = f(\cdot) \tag{7}$$

where $z_i^{(3)}$ is the ith input to a node in Layer 3 and the product is over the inputs of this node. The link weight in Layer 3 $[w_i^{(3)}]$ is then unity. Note that the product operation in the above equation is equivalent to defining a multidimensional (n-dimensional) membership function, which is the

product of the trapezoid functions in (5) over i. This forms a multidimensional trapezoidal membership function called the *hyperbox membership function* [24] since it is defined on a hyperbox in the input space. The corners of the hyperbox are decided by the Layer-2 weights $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ for all i's. More clearly, the interval $[u_{ij}^{(2)}, v_{ij}^{(2)}]$ defines the edge of the hyperbox in the ith dimension. Hence, the weight vector $\{[u_{1j}^{(2)}, v_{1j}^{(2)}], \cdots, [u_{ij}^{(2)}, v_{ij}^{(2)}], \cdots, [u_{nj}^{(2)}, v_{nj}^{(2)}]\}$ defines a hyperbox in the input space. An illustration of a 2-D hyperbox membership function is shown in Fig. 3(b). The rule nodes output are connected to sets of m output-term nodes in Layer 4, one for each output-linguistic variable. This set of output-term nodes defines an m-dimensional trapezoidal (hyperbox) membership function in the output space that specifies the consequent of the rule node. Note that different rule nodes may be connected to the same output hyperbox (i.e., they may have the same consequent), as is shown in Fig. 1.

Layer 4: The nodes in this layer are called output-term nodes; each has two operating modes: down–up transmission and up–down transmission modes (see Fig. 1). In down–up transmission mode, the links in Layer 4 perform the fuzzy OR operation on fired (activated) rule nodes that have the same consequent

$$f[z_i^{(4)}] = \max[z_1^{(4)}, z_2^{(4)}, \cdots, z_p^{(4)}]$$

and

$$a[f(\cdot)] = f(\cdot) \tag{8}$$

where $z_i^{(4)}$ is the ith input to a node in Layer 4 and p is the number of inputs to this node from the rule nodes in Layer 3. Hence, the link weight $w_i^{(4)} = 1$. In up–down transmission mode, the nodes in this layer and the up–down transmission links in Layer 5 function exactly the same as those in Layer 2: each Layer-4 node represents a term of an output-linguistic variable and acts as a 1-D membership function. A set of m output-term nodes (one for each output-linguistic node) defines an m-dimensional hyperbox (membership function) in the output space and there are also two weights $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ on each of the up–down transmission links in Layer 5 (see Fig. 1). The weights define hyperboxes (and, thus, the associated hyperbox membership functions) in the output space. More clearly, the weight vector $\{[u_{1j}^{(5)}, v_{1j}^{(5)}], \cdots, [u_{ij}^{(5)}, v_{ij}^{(5)}], \cdots, [u_{mj}^{(5)}, v_{mj}^{(5)}]\}$, defines a hyperbox in the output space.

Layer 5: Each node in this layer is called a output-linguistic node and corresponds to one output-linguistic variable. There are two kinds of nodes in Layer 5. The first kind of node performs up–down transmission for training data (desired outputs) to feed into the network, acting exactly like the input-linguistic nodes. For this kind of node we have

$$\begin{aligned} f(\overline{y}_i, \overline{y}_i^c) &= (\overline{y}_i, \overline{y}_i^c) \\ &= (\overline{y}_i, 1 - \overline{y}_i) \end{aligned}$$

and

$$a[f(\cdot)] = f(\cdot) \tag{9}$$

where $\overline{y}_i$ is the ith element of the normalized desired output vector. Notice that complement coding is also performed on the desired output vectors. Thus, as mentioned above, there are two weights on each of the up–down transmission links in Layer 5 (the $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ shown in Fig. 1). The weights define hyperboxes and the associated hyperbox membership functions in the output space. The second kind of node performs down–up transmission for decision signal output. These nodes and the Layer-5 down–up transmission links attached to them act as a defuzzifier. If $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ are the corners of the hyperbox of the jth term of the ith output-linguistic variable y_i, then the following functions can be used to simulate the *center of area* defuzzification method:

$$\begin{aligned} f[z_j^{(5)}] &= \sum_j w_{ij}^{(5)} z_j^{(5)} \\ &= \sum_j m_{ij}^{(5)} z_j^{(5)} \end{aligned}$$

and

$$a[f(\cdot)] = \frac{f(\cdot)}{\sum_j z_j^{(5)}} \tag{10}$$

where $z_j^{(5)}$ is the input to the ith output-linguistic node from its jth term node and $m_{ij}^{(5)} = [u_{ij}^{(5)} + v_{ij}^{(5)}]/2$ denotes the center value of the output membership function of the jth term of the ith output-linguistic variable. The center of a fuzzy region is defined as that point with the smallest absolute value among all the other points in the region at which the membership value is equal to one. Here, the weight $w_{ij}^{(5)}$ on a down–up transmission link in Layer 5 is defined by $w_{ij}^{(5)} \equiv m_{ij}^{(5)} = [u_{ij}^{(5)} + v_{ij}^{(5)}]/2$, where $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ are the weights on the corresponding up–down transmission link in Layer 5.

The fuzzy reasoning process in the FALCON is illustrated in Fig. 4, which shows a graphic interpretation of the center of area defuzzification method. Here, we consider a two-input and two-output case. As shown in the figure, three hyperboxes (IH_1, IH_2, and IH_3) are formed in the input space and two hyperboxes (OH_1, OH_2) are formed in the output space. These hyperboxes are defined by the weights u_{ij}, v_{ij}, u'_{ij}, and v'_{ij}. The three fuzzy rules indicated in the figure are "IF $\mathbf{x}$ is IH_1 THEN $\mathbf{y}$ is OH_1 (Rule 1)," "IF $\mathbf{x}$ is IH_2 THEN $\mathbf{y}$ is OH_1 (Rule 2)," and "IF $\mathbf{x}$ is IH_3 THEN $\mathbf{y}$ is OH_2 (Rule 3)," where $\mathbf{x} = (x_1, x_2)$ and $\mathbf{y} = (y_1, y_2)$. If an input pattern is located inside a hyperbox, the membership value is equal to one [see (6)]. In this figure, according to (8), z_1 is obtained by performing fuzzy OR (maximum) operation on the inferred results of Rules 1 and 2, which have the same consequent OH_1. Also according to (8), z_2 is directly the inferred result of Rule 3. z_1 and z_2 are then defuzzified to get the final output according to (10).

Based on the above structure, an on-line learning algorithm FALCON-ART will be proposed to determine the proper corners of the hyperbox (u_{ij}'s and v_{ij}'s) for each term node in layers two and four. Also, it will learn fuzzy logic rules and connection types of the links at layers three and four; that is, the precondition links and consequent links of the rule nodes.

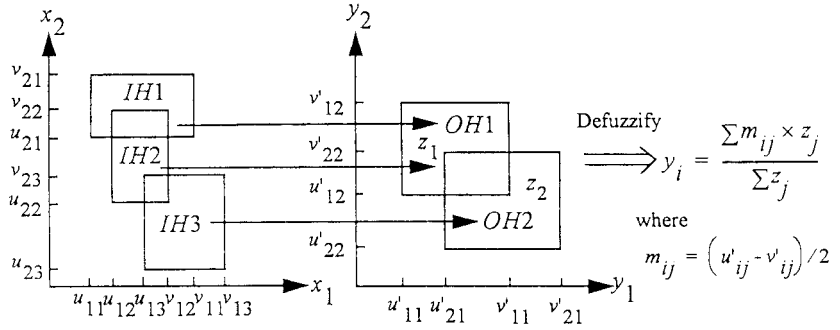


Fig. 4. The fuzzy reasoning process in the FALCON-ART model.

III. The ART-Based Learning Algorithm

In this section, we present an on-line two-step learning scheme for the proposed FALCON-ART model. For an on-line incoming training pattern, the following two steps are performed in this learning scheme. First, a structure learning scheme is used to decide proper fuzzy partitions and to find the presence of rules. Second, a supervised learning scheme is used to optimally adjust the membership functions for the desired outputs. This learning scheme (called FALCON-ART) uses the fast-learn fuzzy ART to perform structure learning and the backpropagation algorithm to perform parameter learning. This structure/parameter learning cycle will be repeated for each on-line incoming training pattern. In this learning method, only the training data need to be provided from the outside world. The users need not provide the initial fuzzy partitions, membership functions, and fuzzy logic rules. Hence, there is no input/output-term nodes and no rule nodes in the beginning of learning. They are created dynamically as learning proceeds upon receiving on-line incoming training data. In other words, an initial form of the network has only input- and output-linguistic nodes before the network is trained. Then, during the learning process new input- and output-term nodes and rule nodes will be added.

A. The Structure Learning Step

The problem for the structure learning can be stated as—given the training input data at time t, $x_i(t)$, $i = 1, \cdots, n$, and the desired output value $y_i(t)$, $i = 1, \cdots, m$, we want to decide proper fuzzy partitions as well as membership functions and find the fuzzy logic rules. In this step, the network works in a two-sided manner; that is, the nodes and links at Layer 4 are in the up–down transmission mode so that the training input and output data are fed into this network from both sides.

The structure-learning step consists of three learning processes: input fuzzy clustering process, output fuzzy clustering process, and mapping process. The first two processes are performed simultaneously on both sides of the network and are described below.

1) Input Fuzzy Clustering Process: We use the fuzzy ART fast-learning algorithm [22], [23] to find the input membership function parameters $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$. This is equivalent to finding proper input-space fuzzy clustering or, more precisely, to forming proper fuzzy hyperboxes in the input space. Initially, for each complement coded input vector $\mathbf{x}'$ [see (1)], the values of choice functions T_j are computed by

$$T_j(\mathbf{x}') = \frac{|\mathbf{x}' \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|}, \qquad j = 1, 2, \cdots, N \tag{11}$$

where "$\wedge$" is the minimum operator performed for the pairwise elements of two vectors, $\alpha \geq 0$ is a constant, N is the current number of rule nodes, and $\mathbf{w}_j$ is the *complement weight vector*, which is defined by $\mathbf{w}_j \equiv \{[u_{1j}^{(2)}, 1 - v_{1j}^{(2)}], \cdots, [u_{ij}^{(2)}, 1 - v_{ij}^{(2)}], \cdots, [u_{nj}^{(2)}, 1 - v_{nj}^{(2)}]\}$. Notice that $\{[u_{1j}^{(2)}, v_{1j}^{(2)}], \cdots, [u_{ij}^{(2)}, v_{ij}^{(2)}], \cdots, [u_{nj}^{(2)}, v_{nj}^{(2)}]\}$ is the weight vector of Layer-2 links associated with rule node j. The choice function value indicates the similarity between the input vector $\mathbf{x}'$ and the complement weight vector $\mathbf{w}_j$. We then need to find the complement weight vector closest to $\mathbf{x}'$. This is equivalent to finding a hyperbox (category) to which $\mathbf{x}'$ could belong. The chosen category is indexed by J where

$$T_J = \max\{T_j : j = 1, \cdots, N\}. \tag{12}$$

Resonance occurs when the match value of the chosen category meets the vigilance criterion

$$\frac{|\mathbf{x}' \wedge \mathbf{w}_J|}{|\mathbf{x}'|} \geq \rho \tag{13}$$

where $\rho \in [0, 1]$ is a vigilance parameter. If the vigilance criterion is not met we say *mismatch reset* occurs. In this case, the choice function value T_J is set to zero for the duration of the input presentation to prevent persistent selection of the same category during search (we call this action "disabling J"). A new index J is then chosen using (12). The search process continues until the chosen J satisfies (13). If no such J is found, then a new input hyperbox is created by adding a set of n new input-term nodes, one for each input-linguistic variable, and setting up links between the newly added input-term nodes and the input-linguistic nodes. The complement weight vectors on these new Layer-2 links are simply given as the current input vector, $\mathbf{x}'$. These newly added input-term nodes and links define a new hyperbox and, thus, a new category in the input space. We denote this newly added hyperbox as J.

2) Output Fuzzy Clustering Process: The output fuzzy clustering process is exactly the same as the input fuzzy clustering process except that it is performed between Layers 4 and 5, which are working in the up–down transmission mode. Of course, the training pattern used now is the desired output

vector after complement coding $\mathbf{y}' = (\overline{y}, \overline{y}^c) = (\overline{y}, 1 - \overline{y})$. We denote the chosen or newly added output hyperbox by K. This hyperbox is defined by the complement weight vector in Layer 5, $\mathbf{w}_K = \{[u_{1j}^{(5)}, 1 - v_{1j}^{(5)}], \cdots, [u_{ij}^{(5)}, 1 - v_{ij}^{(5)}], \cdots, [u_{mj}^{(5)}, 1 - v_{mj}^{(5)}]\}$.

The two fuzzy clustering processes above produce a chosen input hyperbox indexed as J and a chosen output hyperbox indexed as K, where the input hyperbox J is defined by $\mathbf{w}_J$ and the output hyperbox K by $\mathbf{w}_K$. If the chosen input hyperbox J is not newly added, then there is a rule node J that corresponds to it. If the input hyperbox J is a newly added one, then a new rule node (indexed as J) in Layer 3 is added and connected to the input-term nodes that constitute it.

3) Mapping Process: After the two hyperboxes in the input and output spaces are chosen in the input and output fuzzy clustering processes, the next step is to perform the mapping process which decides the connections between Layer-3 and Layer-4 nodes. This is equivalent to deciding the consequents of fuzzy logic rules. This mapping process is described by the following algorithm wherein connecting rule node J to output hyperbox K means connecting the rule node J to the output-term nodes that constitutes the hyperbox K in the output space.

Step 1) IF rule node J is a newly added node THEN connect rule node J to output hyperbox K.

Step 2) ELSE IF rule node J is not connected to output hyperbox K originally THEN disable J and perform input fuzzy clustering process to find the next qualified J [i.e., the next rule node that satisfies (12) and (13)]. Go to Step 1).

Step 3) ELSE no structure change is necessary.

In the mapping process, hyperboxes J and K are resized according to the *fast-learning rule* [22] by updating weights $\mathbf{w}_J$ and $\mathbf{w}_K$ as follows:

$$\begin{aligned} \mathbf{w}_J^{(new)} &= \mathbf{x}' \wedge \mathbf{w}_J^{(old)} \\ \mathbf{w}_K^{(new)} &= \mathbf{y}' \wedge \mathbf{w}_K^{(old)}. \end{aligned} \tag{14}$$

Note that once the consequent of a rule node has been decided in the mapping process, it will not be changed thereafter. We now use Fig. 4 to illustrate the structure-learning step as follows. For a given training datum, the input fuzzy clustering process and the output fuzzy clustering process find or form proper clusters (hyperboxes) in the input and output spaces. Assume that the input and output hyperbox pair found (or formed) are (J, K). The mapping process then tries to relate these two hyperboxes by setting up links between them. This is equivalent to finding a fuzzy logic rule that defines the association between an input hyperbox and an output hyperbox. If this association exists already [e.g., $(J, K) = (IH_1, OH_1)$, (IH_2, OH_1), or (IH_3, OH_2) in Fig. 4], then no structural change is necessary. If input hyperbox J is newly formed and, thus, not connected to any output hyperbox, it is connected to output hyperbox K directly. Otherwise, if input hyperbox J is associated with an output hyperbox different from K originally [e.g., $(J, K) = (IH_2, OH_2)$], then a new input hyperbox close to J will be found or formed by performing the input fuzzy clustering process again. This search (called "match tracking") continues until an input hyperbox J' that can be associated with output hyperbox K is found [e.g., $(J', K) = (IH_3, OH_2)$].

In the structure-learning step, the vigilance parameter ρ is an important parameter. The vigilance value is set between zero and one. A low vigilance value leads to the learning of coarse clusters, whereas a high vigilance value leads to the learning of fine clusters. If the vigilance value is equal to zero, all the training data belong to the same fuzzy cluster in the input space or output space; that is, only one cluster is formed in the input and output spaces in this case. If the vigilance value is set to one, every training datum forms one fuzzy cluster in the input or output space. An increase in sensitivity is modeled within the FALCON-ART model by an increase in the vigilance value. With a fixed vigilance value, the fuzzy clusters may grow too many as learning proceeds. To avoid this problem and to increase learning speed, we had better use adaptive (monotonically decreasing) vigilance values. Initially, we use a high vigilance value such that more and fine clusters are formed in the initial stage of learning. The vigilance value is then decreased gradually. As the vigilance value decrease to some extent, mismatch reset will seldom occurs and new cluster will not be created easily.

B. The Parameter Learning Step

After the network structure has been adjusted according to the current training pattern, the network then enters the second learning step to adjust the parameters of the membership functions optimally with the same training pattern. The problem for the parameter learning can be stated as: given the training input data $x_i(t)$, $i = 1, \cdots, n$, the desired output value $y_i(t)$, $i = 1, \cdots, m$, the input and output hyperboxes, and the fuzzy logic rules, we want to adjust the parameters of the membership functions optimally. These hyperboxes and fuzzy logic rules are learned in the structure-learning step. In the parameter learning, the network works in the feedforward manner; that is, the node and links in Layer 4 are in the down–up transmission mode. Basically, the idea of backpropagation algorithm is used for this parameter learning to find the output errors of the node in each layer. Then, these errors are analyzed to perform parameters adjustment. The goal is to minimize the error function

$$E = \tfrac{1}{2}[y(t) - \hat{y}(t)]^2 \tag{15}$$

where $y(t)$ is the desired output and $\hat{y}(t)$ is the current output. For the current training data pair starting at the input nodes, a forward pass is used to compute the activity levels of all the nodes in the network. Then, starting at the output nodes, a backward pass is used to compute $\partial E/\partial y$ for all the hidden nodes. It is noted that in the parameter learning we use only normalized training vectors $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$ rather than the complement coded ones $\mathbf{x}'$ and $\mathbf{y}'$. Assuming that w is the adjustable parameter in a node, the general learning rule used is

$$w(t+1) = w(t) + \eta\left(-\frac{\partial E}{\partial w}\right) \tag{16}$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial f}\frac{\partial f}{\partial w} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial f}\frac{\partial f}{\partial w} \tag{17}$$

where η is the learning rate. To show the learning rules, we derive the rules layer by layer using the hyperbox membership functions with corners u_{ij}'s and v_{ij}'s as the adjustable parameters for these computations. For clarity, we consider the single output case.

Layer 5: Using (10), (16), and (17), the updating rule of the corners of hyperbox membership function v_i is derived as

$$\frac{\partial E}{\partial v_i} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial v_i} = -[y(t) - \hat{y}(t)]\frac{z_i}{2\sum z_i}. \tag{18}$$

Hence, the corner parameter is updated by

$$v_i(t+1) = v_i(t) + \eta[y(t) - \hat{y}(t)]\frac{z_i}{2\sum z_i}. \tag{19}$$

Similarly, using (10), (16), and (17), the updating rule of the other corner parameter u_i is derived as

$$\frac{\partial E}{\partial u_i} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial u_i} = -[y(t) - \hat{y}(t)]\frac{z_i}{2\sum z_i}. \tag{20}$$

Hence, the other corner parameter is updated by

$$u_i(t+1) = u_i(t) + \eta[y(t) - \hat{y}(t)]\frac{z_i}{2\sum z_i}. \tag{21}$$

The error to be propagated to the preceding layer is

$$\delta^{(5)} = -\frac{\partial E}{\partial a^{(5)}} = y(t) - \hat{y}(t). \tag{22}$$

Layer 4: In the down–up transmission mode, there is no parameter to be adjusted in this layer. Only the error signal $[\delta_i^{(4)}]$ needs to be computed and propagated. According to (10), the error signal $\delta_i^{(4)}$ is derived as in the following:

$$\delta_i^{(4)} = -\frac{\partial E}{\partial a^{(4)}} = -\frac{\partial E}{\partial a^{(5)}}\frac{\partial a^{(5)}}{\partial a^{(4)}} \tag{23}$$

where

$$\frac{\partial E}{\partial a^{(4)}} = -\delta_i^{(5)} \tag{24}$$

$$\frac{\partial a^{(5)}}{\partial a^{(4)}} = \frac{m_i\sum z_i - \sum m_i z_i}{\left(\sum z_i\right)^2}. \tag{25}$$

Hence, the error signal is

$$\delta_i^{(4)} = \delta^{(5)}\frac{m_i\sum z_i - \sum m_i z_i}{\left(\sum z_i\right)^2}. \tag{26}$$

In the multiple output case, the computations in Layers 4 and 5 are exactly the same as the above and proceed independently for each output-linguistic variable.

Layer 3: As in Layer 4, only the error signals need to be computed. According to (8), this error signal can be derived as

$$\delta_i^{(3)} = -\frac{\partial E}{\partial a^{(3)}} = -\frac{\partial E}{\partial a^{(4)}}\frac{\partial a^{(4)}}{\partial f^{(4)}}\frac{\partial f^{(4)}}{\partial a^{(3)}} \tag{27}$$

where

$$\frac{\partial E}{\partial a^{(4)}} = -\delta_i^{(4)} \tag{28}$$

$$\frac{\partial a^{(4)}}{\partial f^{(4)}} = 1 \tag{29}$$

$$\frac{\partial f^{(4)}}{\partial a^{(3)}} = \frac{\partial f^{(4)}}{\partial z_i^{(4)}} = \frac{z_i^{(4)}}{z_{\max}} \tag{30}$$

where $z_{\max} = \max(\text{inputs of output-term nodes } j)$. The term, $z_i^{(4)}/z_{\max}$, is to normalize the error to be propagated for the fired rules with the same consequent. Hence, the error signal is

$$\delta_i^{(3)} = \frac{z_i^{(4)}}{z_{\max}}\delta_i^{(4)}. \tag{31}$$

If there are multiple outputs, then the error signal becomes $\delta_i^{(3)} = \sum_k [z_k^{(4)}/z_{\max}]\delta_k^{(4)}$, where the summation is performed over the consequents of a rule node; that is, the error of a rule node is the summation of the errors of its consequents.

Layer 2: Using (5), (16), and (17), the updating rule of v_{ij} is derived as in the following:

$$-\frac{\partial E}{\partial v_{ij}} = -\frac{\partial E}{\partial a^{(3)}}\frac{\partial a^{(3)}}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial v_{ij}} \tag{32}$$

where

$$\frac{\partial a^{(3)}}{\partial a^{(2)}} = \prod_{k\neq i} z_k^{(3)} \tag{33}$$

$$\frac{\partial a^{(2)}}{\partial v_{ij}} = \begin{cases}\gamma, & \text{if } 0 \le (x_i - v_{ij})\gamma \le 1\\ 0, & \text{otherwise.}\end{cases} \tag{34}$$

So the updating rule of v_{ij} is

$$v_{ij}(t+1) = v_{ij}(t) + \eta\frac{\partial a^{(2)}}{\partial v_{ij}}\delta_i^{(3)}\prod_{k\neq i} z_k^{(3)}. \tag{35}$$

Similarly, using (5), (16), and (17), the updating rule of u_{ij} is derived as

$$-\frac{\partial E}{\partial u_{ij}} = -\frac{\partial E}{\partial a^{(3)}}\frac{\partial a^{(3)}}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial u_{ij}} \tag{36}$$

where

$$\frac{\partial a^{(3)}}{\partial a^{(2)}} = \prod_{k\neq i} z_k^{(3)} \tag{37}$$

$$\frac{\partial a^{(2)}}{\partial u_{ij}} = \begin{cases}-\gamma & \text{if } 0 \le (u_{ij} - x_i)\gamma \le 1\\ 0 & \text{otherwise.}\end{cases} \tag{38}$$

Hence, the updating rule of u_{ij} becomes

$$u_{ij}(t+1) = u_{ij}(t) + \eta \frac{\partial a^{(2)}}{\partial u_{ij}} \delta_i^{(3)} \prod_{k \neq i} z_k^{(3)}. \quad (39)$$

IV. Rule Annihilation

With the above on-line learning algorithm, FALCON-ART, the fuzzy logic rules can only be created and cannot be annihilated. It would be better if the FALCON-ART has the ability to delete unnecessary or redundant rules. In this section, we shall present two methods of rule annihilation. The basic idea is to combine two or more "similar" rules into a representative one. During the on-line structure/parameter learning, the hyperboxes of input and output spaces are tuned. These hyperboxes may be updated to greater hyperboxes or smaller hyperboxes gradually. Thus, the perfect inclusion between two hyperboxes may happen. Let's consider the situation that the hyperbox j contains the hyperbox k; i.e., the hyperbox k is a subset of the hyperbox j. In this situation, it is intuitive that one of the hyperboxes (j or k) can be annihilated. This is equivalent to the case that two fuzzy terms of a linguistic variable represent redundant fuzzy meaning and thus one of them can be removed.

To determine the rule similarity, a dimension-by-dimension comparison process of hyperboxes is conducted. Let $[u_{ij}, v_{ij}]$ and $[u_{ik}, v_{ik}]$ be the edges of two compared hyperboxes j and k in the ith dimension. If $u_{ij} \leq u_{ik}$ and $v_{ij} \geq v_{ik}$ for all dimensions $i = 1, 2, \cdots$, in the input space or output space we say the hyperbox j contains the hyperbox k. For such two hyperboxes j and k, if we annihilate the hyperbox k and remain the hyperbox j we will obtain the greater hyperbox. This will cause the membership functions to do coarse clustering in the input or output space. On the contrary, if the hyperbox j is annihilated and the hyperbox k remains, we will obtain the smaller hyperbox and the membership functions will do finer clustering in the input or output space. In the latter case, the training patterns outside the hyperbox k will be reclustered in the next iteration. When we decide to annihilate an input hyperbox, we delete all the nodes and attached connections that constitute this hyperbox. We also need to annihilate the output hyperbox associated by this input hyperbox if this output byperbox is not associated by other input byperboxes. On the other hand, if we decide to annihilate a output byperbox, we delete all the nodes that constitute this hyperbox, and then redirect all its input links to the remaining similar output hyperbox. The proposed rule annihilation process is based on the natural property of most control systems that if two input data are close (similar) in the input space, the two mapped outputs are also close (similar) in the output space. Based on the above discussion, we propose two methods to determine the combination of two similar rules. The first method is to keep the greater hyperbox in the input and output spaces, whereas the second method is to keep the smaller hyperbox in the input and output spaces. These two methods are described as follows. The rule annihilation process is illustrated in Figs. 5 and 6, where we consider two-input and two-output case.

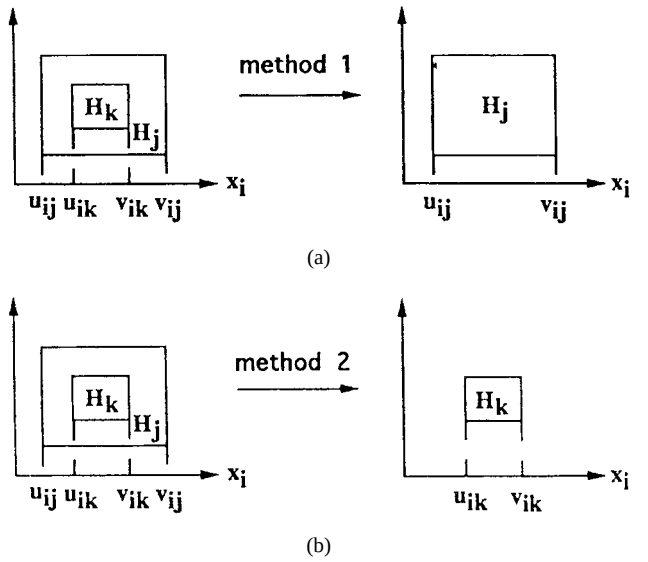


Fig. 5. Two rule-annihilation methods.

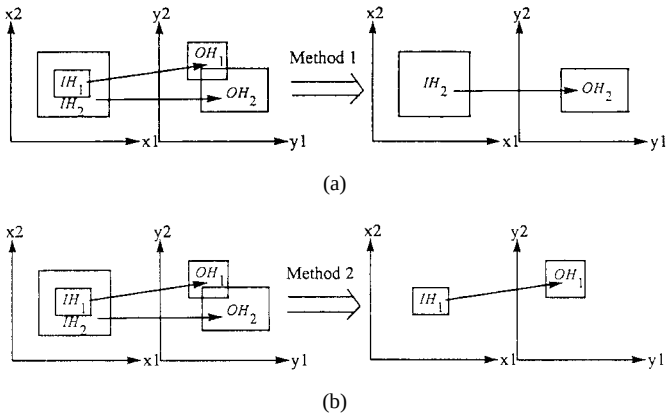


Fig. 6. Illustrations of the rule-annihilation process in the FALCON-ART model.

1) Method 1: If one hyperbox j contains another hyperbox k (i.e., $IH_j \supset IH_k$ or $OH_j \supset OH_k$), we can delete the smaller hyperbox k (i.e., IH_k or OH_k) and keep the greater hyperbox j (i.e., IH_j or OH_j). This is achieved by letting the left-flat points (u_{ij} and u_{ik}) perform the fuzzy AND operation for all dimensions and the right-flat points (v_{ij} and v_{ik}) perform the fuzzy OR operation for all dimensions

$$u_{ij} \wedge u_{ik} = \min(u_{ij}, u_{ik}) = u_{ij} \quad (40)$$

$$v_{ij} \vee v_{ik} = \max(v_{ij}, v_{ik}) = v_{ij}. \quad (41)$$

From these operations we can obtain the greater hyperbox in the input and output spaces [Fig. 5(a)].

The rule-annihilation process in the FALCON-ART is illustrated in Fig. 6, which shows a graphic interpretation of Methods 1 and 2 of the rule-annihilation process. Here, we consider a two-input and two-output case. As shown in the figure, two hyperboxes (IH_1, IH_2) are formed in the input space and two hyperboxes (OH_1, OH_2) are formed in the

output space. The two fuzzy rules indicated in the figure are "IF $\mathbf{x}$ is IH_1 THEN $\mathbf{y}$ is OH_1 (Rule 1)," and "IF $\mathbf{x}$ is IH_2 THEN $\mathbf{y}$ is OH_2 (Rule 2)," where $\mathbf{x} = (x_1, x_2)$ and $\mathbf{y} = (y_1, y_2)$. These rules reflect the natural property of most control problems that if two input data are close (similar) in the input space, the two mapped outputs are also close in the output space. In Fig. 6(a), the hyperbox IH_2 contains the other hyperbox IH_1. According to the Method 1 of rule-annihilation process, we delete the smaller hyperbox IH_1 and keep the greater hyperbox IH_2; that is, we combine Rules 1 and 2 into Rule 2 as shown in Fig. 6(a).

2) Method 2: If one hyperbox j contains another hyperbox k (i.e., $IH_j \supset IH_k$ or $OH_j \supset OH_k$), we can delete the greater hyperbox j (i.e., IH_j or OH_j) and keep the smaller hyperbox k (i.e., IH_k or OH_k). This is achieved by letting the left-flat points (u_{ij} and u_{ik}) perform the fuzzy OR operation for all dimensions and the right-flat points (v_{ij} and v_{ik}) perform the fuzzy AND operation for all dimensions

$$u_{ij} \vee u_{ik} = \max(u_{ij}, u_{ik}) = u_{ik} \tag{42}$$

$$v_{ij} \wedge v_{ik} = \min(v_{ij}, v_{ik}) = v_{ik}. \tag{43}$$

From these operations we can obtain the smaller hyperbox in the input and output spaces [Fig. 5(b)].

We also use Fig. 6 as an example to explain the Method 2 of rule-annihilation process. In Fig. 6(b), the hyperbox IH_2 contains the other hyperbox IH_1. According to Method 2 of rule-annihilation process, we delete the greater hyperbox IH_2 and keep the smaller hyperbox IH_1; that is, we combine Rules 1 and 2 into Rule 1, as shown in Fig. 6(b).

V. Illustrative Examples

A general purpose simulator for the FALCON-ART model has been written in "C" language and runs on a PC-486. Using this simulator, four typical examples are presented in this section to show the fundamental applications of the proposed model. The first example is to identify a dynamic system [25], the second example is to predict time-series [17], the third example is to control the truck backer-upper [26], [27], and the fourth example is to control the ball and beam system [16], [28].

Example 1—Identification of the Dynamic System: In this example, the proposed FALCON-ART model is used to identify a dynamic system. The identification model has the form

$$\hat{y}(k+1) = \hat{f}[u(k), u(k-1), \cdots, u(k-p+1) \; y(k), y(k-1), \cdots, y(k-q+1)]. \tag{44}$$

Since both the unknown plant and the FALCON-ART model are driven by the same input, the FALCON-ART model adjusts itself with the goal of causing the output of the identification model to match that of the unknown plant. Upon convergence, the input/output response relationship should match.

The plant to be identified is guided by the difference equation

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k). \tag{45}$$

The output of the plant depends nonlinearly on both its past output values and the input values, but the effects of the input and output values are additive. In applying the FALCON-ART model to this identification problem, the learning rate $\eta = 0.005$, sensitivity parameter $\gamma = 4$, and vigilance parameter $\rho_{\text{input}} = 0.6$, $\rho_{\text{output}} = 0.5$ are chosen, where ρ_{input} and ρ_{output} are vigilance parameters used in the input and output fuzzy clustering processes, respectively. The training input patterns are generated with $u(k) = \sin(2\pi k/100)$ and the training process is continued for 60 000 time steps. Starting at zero, the number of clusters grow dynamically for incoming training data. Each cluster corresponds to a hyperbox in the input or output space. Fig. 7 shows the root-mean-square (rms) errors during learning. Each point on the curve is the average of 200 training time steps. The curve appears to have a big oscillation at the beginning of learning. This situation reflects the structure changing in the early stage of learning; that is, the numbers of fuzzy partitions of x_1, x_2, and y_1 are increasing and new fuzzy logic rules are generated. In this example, the case of two input and one output is considered for illustration. Fig. 8(a) illustrates the distribution of the training patterns and the final assignment of the rules (i.e., distribution of the membership functions) in $[u(k), y(k)]$ plain (input space). There are six hyperboxes (IH_1, IH_2, IH_3, IH_4, IH_5, IH_6) formed in the input space. Fig. 8(b) shows the distribution of the output membership functions in $y(k+1)$ domain (output space). Three trapezoidal membership functions (OH_1, OH_2, OH_3) are generated in the output space. After the hyperboxes in the input and output spaces are tuned or created in the fuzzy clustering process, the mapping process then decides proper mapping between the input clusters and output clusters. There are six fuzzy logic rules formed, finally, as in the following:

Rule 1: IF $\mathbf{x}$ is IH_1, THEN $\mathbf{y}$ is OH_3
Rule 2: IF $\mathbf{x}$ is IH_2, THEN $\mathbf{y}$ is OH_1
Rule 3: IF $\mathbf{x}$ is IH_3, THEN $\mathbf{y}$ is OH_3
Rule 4: IF $\mathbf{x}$ is IH_4, THEN $\mathbf{y}$ is OH_2
Rule 5: IF $\mathbf{x}$ is IH_5, THEN $\mathbf{y}$ is OH_1
Rule 6: IF $\mathbf{x}$ is IH_6, THEN $\mathbf{y}$ is OH_2,

where $\mathbf{x} = [u(k), y(k)]$ and $\mathbf{y} = y(k+1)$. From these fuzzy logic rules, we know Rules 1 and 3 have the same consequent. Also, Rules 2 and 5 and Rules 4 and 6 map to the same consequent. These rules reflect the natural property that if two input data are close (similar) in the input space, the two mapped outputs are also close in the output space. Fig. 9 shows the outputs of the plant and the identification model. In this figure, the output of the FALCON-ART model are presented as dotted curve while plant output values are represented as solid curve. The results show the perfect identification capability of the trained FALCON-ART model.

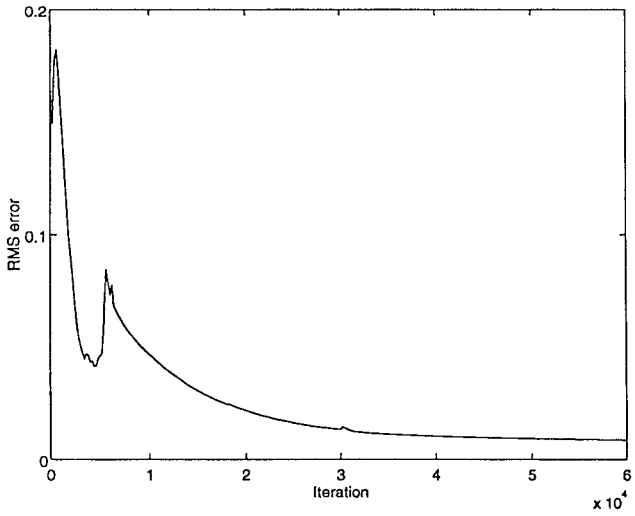
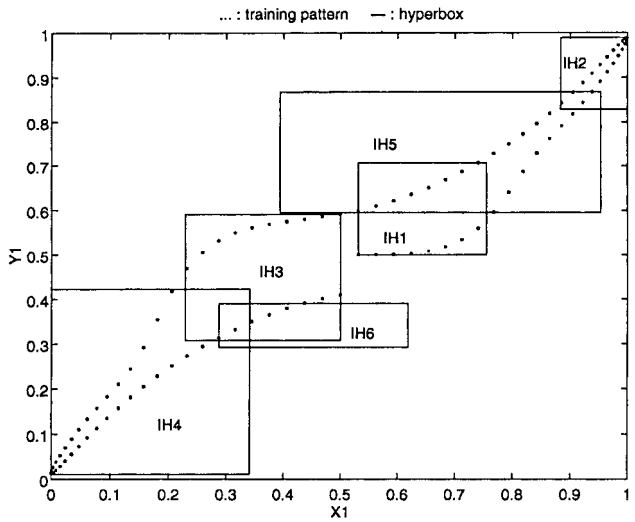
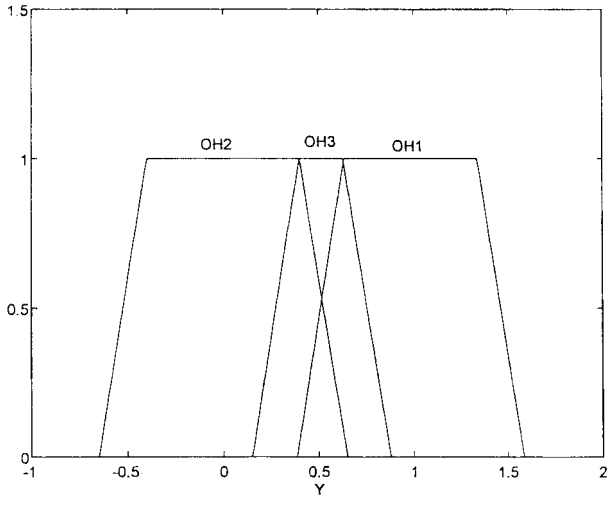


Fig. 7. Learning curve of the FALCON-ART identifier in Example 1.



(a)



(b)

Fig. 8. Simulation results of the FALCON-ART model without rule annihilation in Example 1. (a) The input training patterns and the final assignment of rules. (b) The distribution of output membership functions.

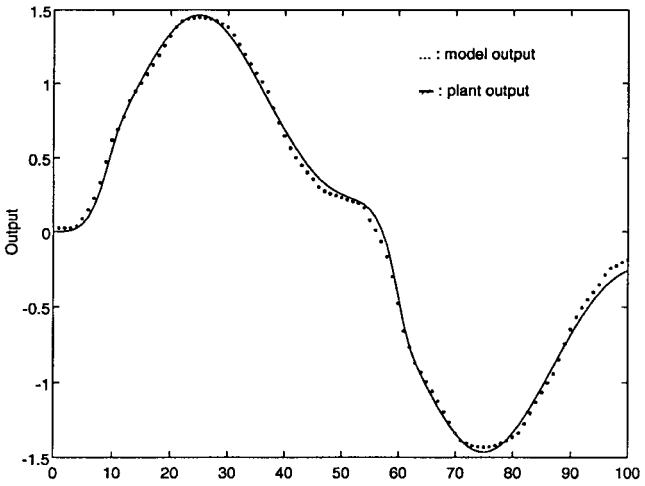


Fig. 9. The outputs of the plant and the identification model.

During the simulation, we also applied the rule-annihilation techniques into the FALCON-ART. The learning results of the FALCON-ART with the Method 1 of rule annihilation are shown in Fig. 10. There are four fuzzy logic rules generated in this case. Fig. 10(a) and (b) show the distribution of the input and output membership functions, respectively. The four generated fuzzy logic rules are "IF $\mathbf{x}$ is IH_1 THEN $\mathbf{y}$ is OH_3," "IF $\mathbf{x}$ is IH_2 THEN $\mathbf{y}$ is OH_2," "IF $\mathbf{x}$ is IH_3 THEN $\mathbf{y}$ is OH_4," and "IF $\mathbf{x}$ is IH_4 THEN $\mathbf{y}$ is OH_1." If we use Method 2 of rule annihilation, there are five fuzzy logic rules generated when learning is terminated. Fig. 11(a) and (b) shows the distribution of the input and output membership functions in this case. The five generated fuzzy logic rules connecting these two figures are "IF $\mathbf{x}$ is IH_1 THEN $\mathbf{y}$ is OH_3," "IF $\mathbf{x}$ is IH_2 THEN $\mathbf{y}$ is OH_2," "IF $\mathbf{x}$ is IH_3 THEN $\mathbf{y}$ is OH_1," "IF $\mathbf{x}$ is IH_4 THEN $\mathbf{y}$ is OH_1," and "IF $\mathbf{x}$ is IH_5 THEN $\mathbf{y}$ is OH_3." Simulation results show that with rule-annihilation techniques, we can still obtain perfect identification capability by using fewer fuzzy logic rules.

Notice that the possibility of the occurrence of the perfect inclusion between two hyperboxes is not low. This is the nature of the fuzzy ART and fuzzy ARTMAP algorithms [22], [23]. Especially, when a large hyperbox formed in the input (output) space, a following input pattern may easily fall into it. If this pattern is not considered to belong to the large hyperbox (e.g., the desired output of this pattern is not in the output hyperbox that the large input hyperbox associates with), it will form a new small hyperbox in the large hyperbox and perfect inclusion occurs. Moreover, since our model uses the fast-learn fuzzy ART to decide the initial hyperboxes (structure learning) and the backpropagation algorithm to tune the hyperboxes (parameter learning), a hyperbox may be tuned to include or be included by another hyperbox gradually in the parameter learning step and perfect inclusion between two hyperboxes may happen.

Example 2—Prediction of the Chaotic Time-Series: Let $p(k)$, $k = 1, 2, \cdots$, be a time series. The problem of time-series prediction can be formulated as: given $p(k-m+1), p(k-m+2), \cdots, p(k)$, determine $p(k+l)$, where m and l are fixed positive integer (i.e., determine a mapping from

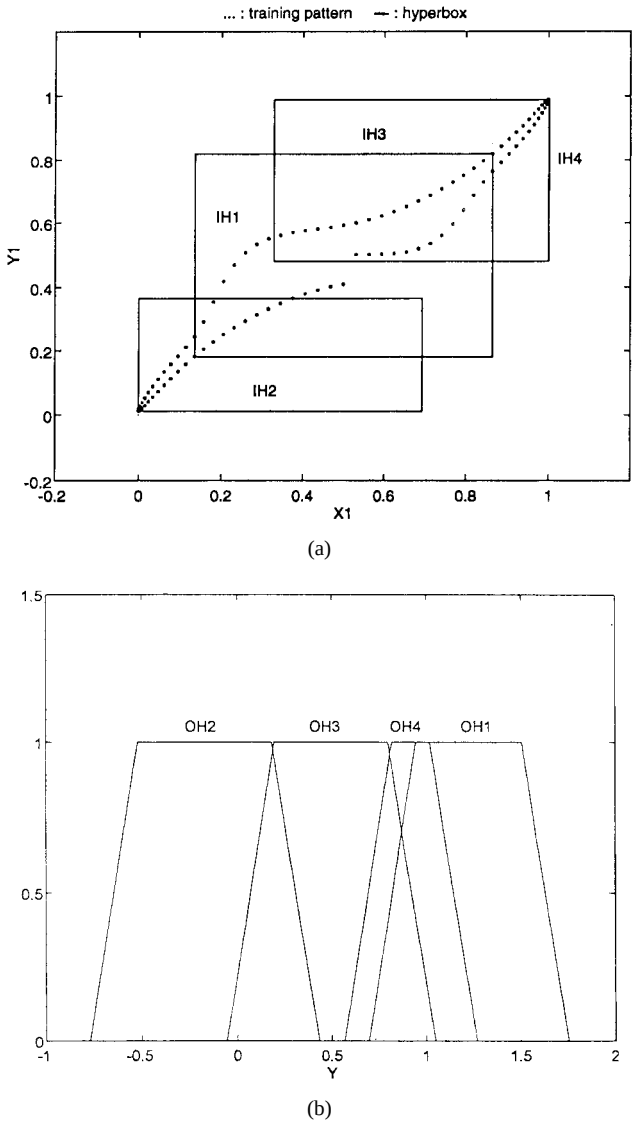


Fig. 10. Simulation results of the FALCON-ART model with Method 1 of rule annihilation in Example 1. (a) The input training patterns and the final assignment of rules. (b) The distribution of output membership functions.

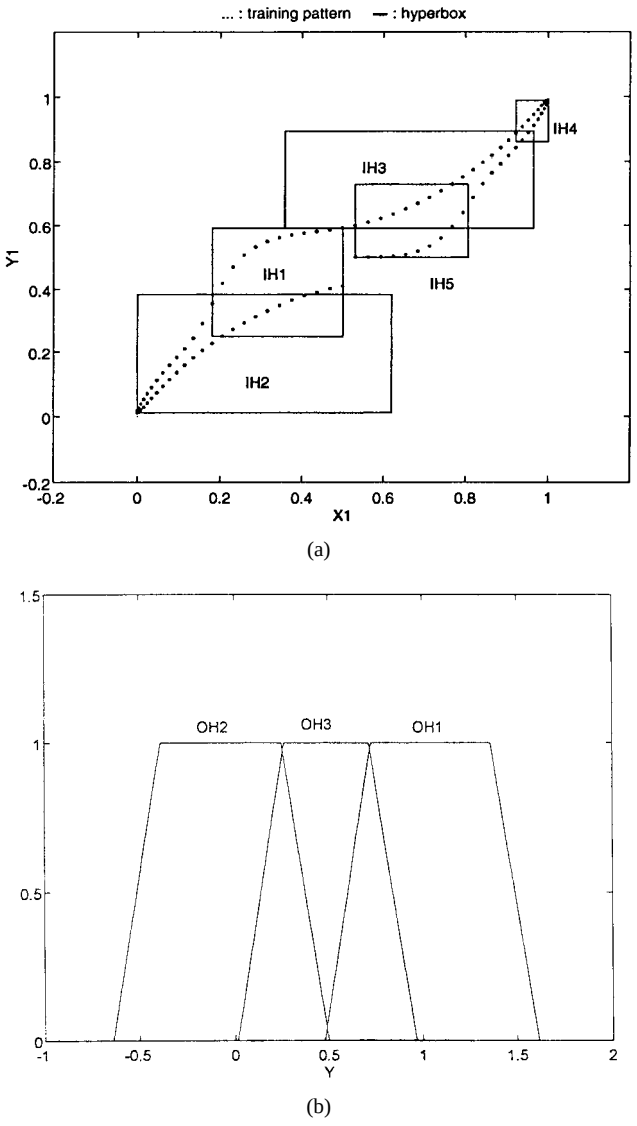


Fig. 11. Simulation results of the FALCON-ART model with Method 2 of rule annihilation in Example 1. (a) The input training patterns and the final assignment of rules. (b) The distribution of output membership functions.

$[p(k-m+1), p(k-m+2), \cdots, p(k)] \in \Re^m$ to $[p(k+l)] \in \Re$). To illustrate the on-line learning ability, the FALCON-ART model is used to predict the Mackey–Glass chaotic time-series. The Mackey–Glass chaotic time-series is generated from the following delay differential equation:

$$\frac{dx(t)}{dt} = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{46}$$

where $\tau > 17$. In our simulation, we choose the series with $\tau = 30$. Fig. 12 shows 1000 points of this chaotic series used to test the FALCON-ART model. We choose $m = 9$ and $l = 1$ in our simulation (i.e., nine point values in the series are used to predict the value of the next time point). The 200 points of the series from $x(501)$–$x(700)$ are used as training data, and the final 300 points from $x(701)$–$x(1000)$ are used as test data. The learning rate $\eta = 0.005$, sensitivity parameter $\gamma = 4$, and vigilance parameter $\rho_{\text{input}} = 0.6$, $\rho_{\text{output}} = 0.5$ are chosen, where ρ_{input} and ρ_{output} are vigilance parameters used in the input and output fuzzy clustering processes, respectively. After the structure-parameter learning, there are twenty-two fuzzy logic rules generated in our model. Fig. 13 shows the prediction of the chaotic time series from $x(701)$–$x(1000)$ when 200 training data [from $x(501)$–$x(700)$] are used. In this figure, predictions of the FALCON-ART model are represented as o's while true values are represented as *'s. The rms error of prediction output approximates 0.08. The results show the good prediction capability of the FALCON-ART model trained only by a small set of training data.

We now compare the performance of our system with that of other existing methods that can generate fuzzy rules from numerical data automatically. The performance indexes considered include numbers of fuzzy rules generated and rms error of prediction output. The comparison results are tabulated in Table I. First, we compare the performance of the FALCON-ART model with that of the system proposed by Wang and Mendel [17]. They developed a general method to generate fuzzy rules from numerical data. This method consists of five steps: Step 1 divides the input and output spaces

TABLE I
PERFORMANCE COMPARISON OF VARIOUS RULE GENERATION METHODS ON THE TIME-SERIES PREDICTION PROBLEM

	FALCON-ART		Wang & Mendel	Data distribution	Kosko (AVQ) without backpropagation		Kosko (AVQ) with backpropagation			
					UCL	DCL	UCL		DCL	
Rule number (200 training data)	22	30*	121	118	100	100	22	100	22	100
RMS error	0.08	0.04	0.08	0.08	0.17	0.2	0.16	0.09	0.17	0.09

* 700 training data are used

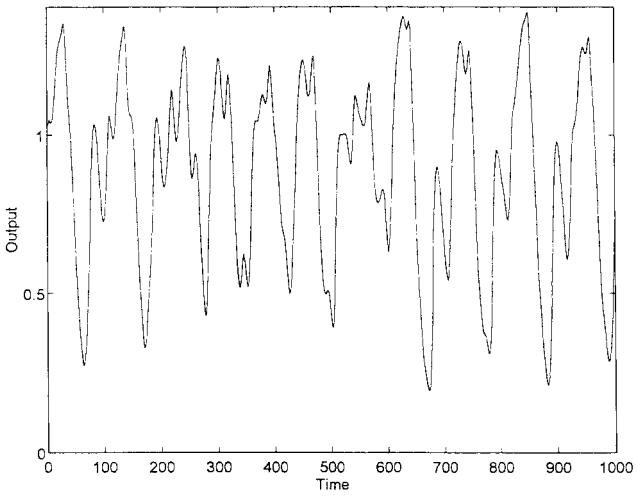


Fig. 12. The Mackey–Glass chaotic time series.

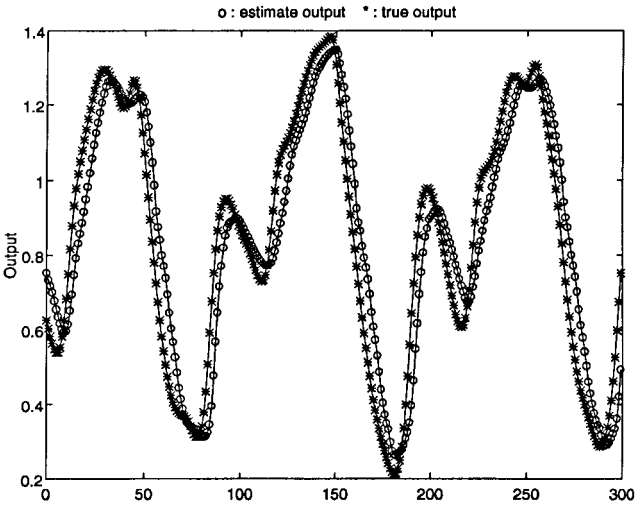


Fig. 13. Simulation results of the time-series prediction from $x(701)$–$x(1000)$ using the FALCON-ART model with 22 fuzzy rules when 200 training data [from $x(501)$–$x(700)$] are used.

of the given numerical data into fuzzy regions by choosing proper membership functions, Step 2 generates fuzzy rules from the given data, Step 3 assigns a degree of each of the generated rules for the purpose of resolving conflicts among the generated rules, Step 4 creates a combined fuzzy rule base based on both the generated rules and linguistic rules of human experts, and Step 5 determines a mapping from input space to output space based on the combined fuzzy rule base using a

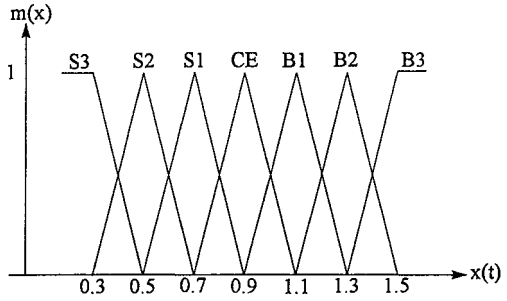


Fig. 14. The membership functions used in Wang and Mendel's system for the chaotic time-series prediction.

defuzzifying procedure. They also chose $m = 9$ and $l = 1$ in their simulation, i.e., nine point values in the series were used to predict the value of the next time point. The membership functions used in the system are shown in Fig. 14. The 200 points of the series in Fig. 12 from $x(501)$–$x(700)$ are used as training data and the final 300 points from $x(701)$–$x(1000)$ are used as test data. After performing the five steps described in the above, we have 121 generated fuzzy rules. Fig. 15 shows the prediction of the chaotic time series from $x(701)$–$x(1000)$ when 200 training data [from $x(501)$–$x(700)$] are used. The rms error of prediction output approximates 0.08. The results show that the proposed FALCON-ART predictor is able to keep similar rms error but needs fewer fuzzy rules than Wang and Mendel's system. These results are shown in the second column of Table I.

The second rule-generation method for comparison is called the data-distribution method. This method generates fuzzy rules according to the training data distribution in the input/output product space. This method consists of five steps: Step 1 divides the input/output product space of the given numerical data into crisp regions, Step 2 generates fuzzy rules from numerical data allocated in the regions and then decides the weight of each rule according to the number of numerical data falling into each region, Step 3 fuzzifies these crisp regions into fuzzy regions by defining proper membership functions, Step 4 creates a combined fuzzy rule base based on the generated rules, and Step 5 determines a mapping from input space to output space based on the combined fuzzy rule base using a defuzzifying procedure. The membership functions shown in Fig. 14 are also used in the data distribution method for chaotic time series prediction. After performing the five steps stated in the above, we have 118 generated fuzzy rules. Fig. 16 shows the prediction of the chaotic time

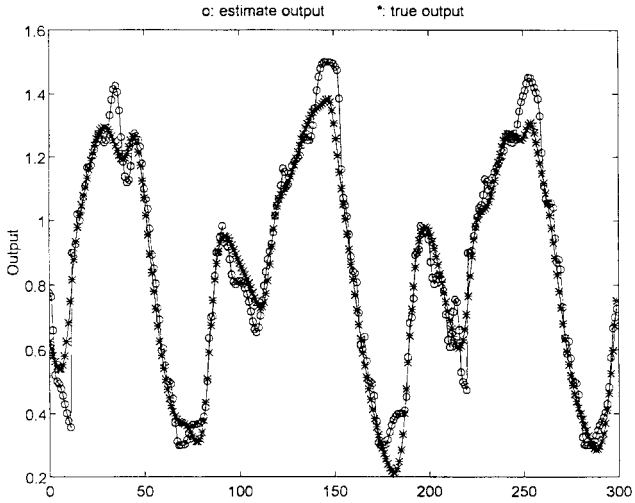


Fig. 15. Simulation results of the time series prediction from $x(701)$–$x(1000)$ using the Wang and Mendel's system when 200 training data [from $x(501)$–$x(700)$] are used.

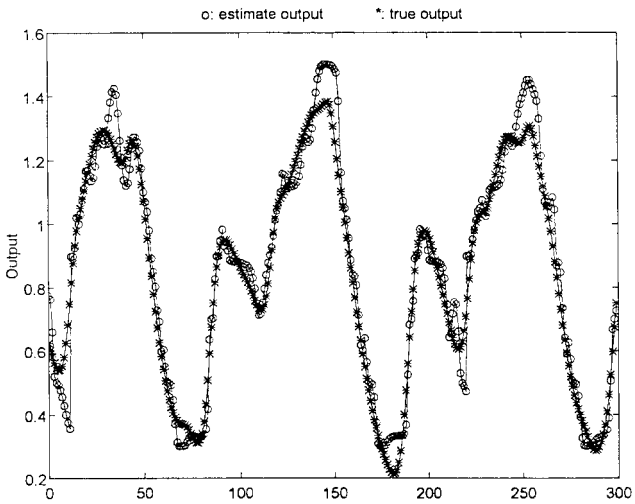


Fig. 16. Simulation results of the time series prediction from $x(701)$–$x(1000)$ using the data distribution method when 200 training data [from $x(501)$–$x(700)$] are used.

series from $x(701)$–$x(1000)$ when 200 training data are used. The rms error of prediction output approximates 0.08. These results are shown in the third column of Table I. According to these results, the number of fuzzy rules generated in our FALCON-ART model is still much smaller than that of the data distribution method. It is noted that the number of the generated fuzzy rules in the data distribution method and Wang and Mendel's system cannot be determined by the users. That is, the number of fuzzy rule generated from the data distribution method and Wang and Mendel's system depends on incoming training patterns.

Kosko [14] proposed the product space-clustering technique to the adaptive fuzzy associative memory (FAM) systems such that they can generate FAM rules directly from training data. The basic concept of automatic generation of FAM rules is to use the adaptive vector quantization (AVQ) algorithm to find and allocate synaptic quantization vectors to FAM cell from input/output training data, and then decide the weight

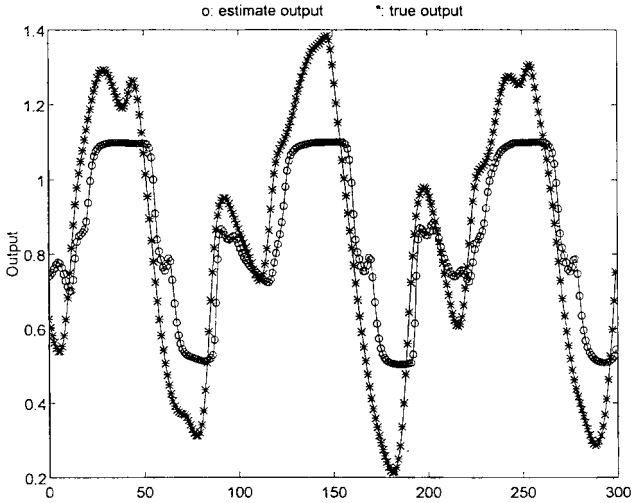


Fig. 17. Simulation results of the time series prediction from $x(701)$–$x(1000)$ using the UCL–AVQ algorithm with 100 fuzzy rules when 200 training data [from $x(501)$–$x(700)$] are used.

of each FAM cell (FAM rule) according to the number of quantization vectors falling into it. Consider the case that $X = \Re$ and $Y = \Re$. The system samples the nonfuzzy input and output stream $(x_1, y_1), (x_2, y_2), \cdots$. Competitive AVQ algorithm, which includes unsupervised competitive learning (UCL) and differential competitive learning (DCL), distributes the k synaptic quantization vectors to different FAM cells F_{ij} in the $X \times Y$ space, where k is an integer given in advance. To do this, we use a UCL or DCL network with two input nodes and k output (competitive) nodes. Assume F_{ij} contains k_{ij} quantization vectors. Then cell counts k_{ij} define a frequency histogram, since all k_{ij} sum to k. Hence, $w_{ij} = k_{ij}/k$ weights the corresponding FAM rule. According to the desired number of fuzzy rules or a given rule–weight threshold, a final FAM rule base is generated.

The membership functions in Fig. 14 are also used in the product space-clustering method for the chaotic time-series prediction problem. In our simulation, after competitive AVQ learning we find that the centroid defuzzification process cannot be performed on the used triangular membership functions since some test data do not fire any fuzzy rules in the rule base and thus causes a "divide by zero" problem in the defuzzification process. To solve this problem, we replace triangular membership functions with bell-shaped membership functions. We set the number of rules m as 100 and use the fixed weighting of rule w_{ij} calculated from cell count k_{ij} after competitive AVQ learning. Figs. 17 and 18 show the prediction of the chaotic time series from $x(701)$–$x(1000)$ using UCL–AVQ and DCL–AVQ algorithms, respectively. The rms errors of prediction output approximate 0.17 and 0.2.

To improve the prediction accuracy, we combine UCL(DCL)–AVQ algorithm with the backpropagation learning algorithm. The former determines the fuzzy rule base and the latter tunes the weighting values of fuzzy rules (w_{ij}). For comparison, we consider two cases: $m = 22$ and $m = 100$. Figs. 19 and 20 show the prediction of the chaotic time series from $x(701)$–$x(1000)$ using the UCL-AVQ and backpropagation hybrid learning algorithm. Figs. 21 and

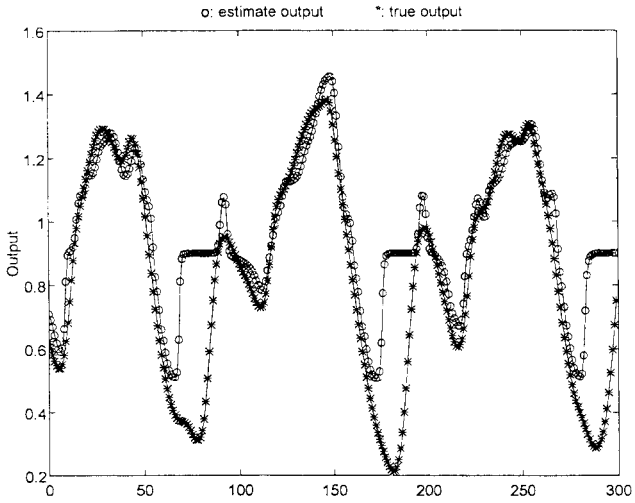


Fig. 18. Simulation results of the time series prediction from $x(701)$–$x(1000)$ using the DCL–AVQ algorithm with 100 fuzzy rules when 200 training data [from $x(501)$–$x(700)$] are used.

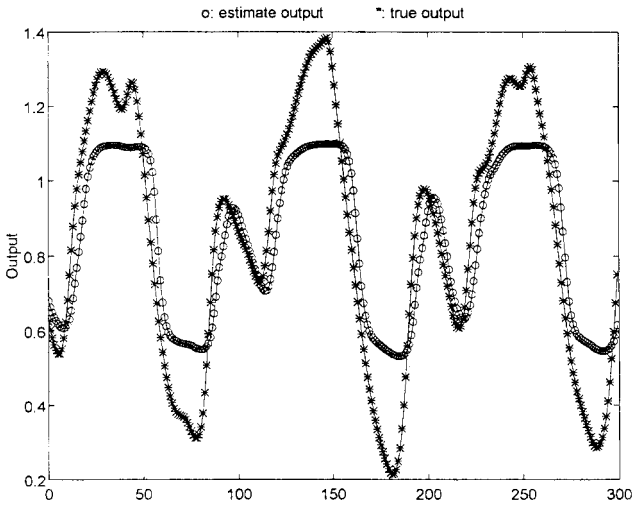


Fig. 19. Simulation results of the time-series prediction from $x(701)$–$x(1000)$ using the UCL–AVQ and backpropagation hybrid learning algorithm with 22 fuzzy rules when 200 training data [from $x(501)$–$x(700)$] are used.

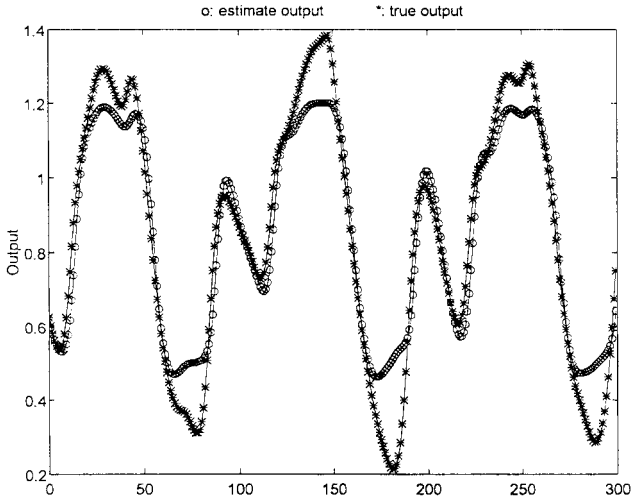


Fig. 20. Simulation results of the time-series prediction from $x(701)$–$x(1000)$ using the UCL–AVQ and backpropagation hybrid learning algorithm with 100 fuzzy rules when 200 training data [from $x(501)$–$x(700)$] are used.

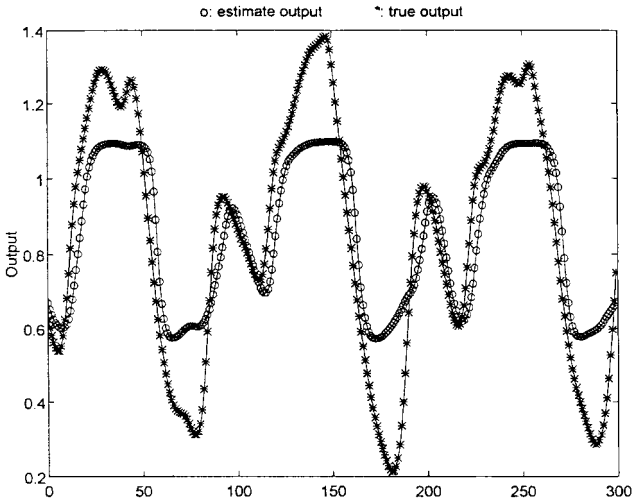


Fig. 21. Simulation results of the time-series prediction from $x(701)$–$x(1000)$ using the DCL–AVQ and backpropagation hybrid learning algorithm with 22 fuzzy rules when 200 training data [from $x(501)$–$x(700)$] are used.

22 show the prediction of the chaotic time series from $x(701)$–$x(1000)$ using the DCL–AVQ and backpropagation hybrid learning algorithm. The results show that the predictor that uses hybrid learning algorithm is able to keep smaller rms error than the original predictor. These results are shown in Table I.

As shown in Table I, the proposed FALCON-ART predictor produces smaller or similar rms error by using fewer fuzzy rules than other rule generation methods. However, these simulation results do not show the perfect prediction capability of the proposed model trained only by a small set of training data. It seems that 200 training data are not enough for the chaotic time-series prediction problem. Thus, we replace 200 training data with 700 training data to train the FALCON-ART predictor. The learning parameters used are the same as those used previously. After the structure-parameter learning, there are 30 fuzzy rules generated and the rms error of prediction output approximates 0.04. Fig. 23 shows the prediction of the chaotic time series from $x(701)$–$x(1000)$ when 700 training data [from $x(1)$–$x(700)$] are used. The simulation results show the perfect prediction capability of the well-trained FALCON-ART. It is noted that the big increase of training data does not induce impractical increase of fuzzy rules in the FALCON-ART model. In [17], Wang and Mendel tried to improve the prediction accuracy by increasing the training data, using the updating fuzzy rule base procedure and dividing the "domain interval" into finer regions in their system. Finally, their system achieved perfect prediction capability when the domain interval was divided into 29 regions. Hence, the price paid for achieving high-prediction accuracy is a larger fuzzy rule base. Recently, Jang [9] proposed a model, called adaptive-network-based fuzzy inference system (ANFIS) architecture, for learning and tuning a fuzzy predictor. By using a hybrid

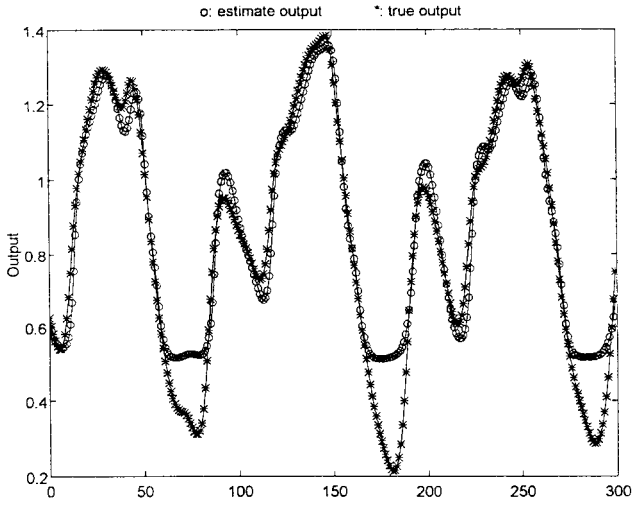


Fig. 22. Simulation results of the time-series prediction from $x(701)$–$x(1000)$ using the DCL–AVQ and backpropagation hybrid learning algorithm with 100 fuzzy rules when 200 training data [from $x(501)$–$x(700)$] are used.

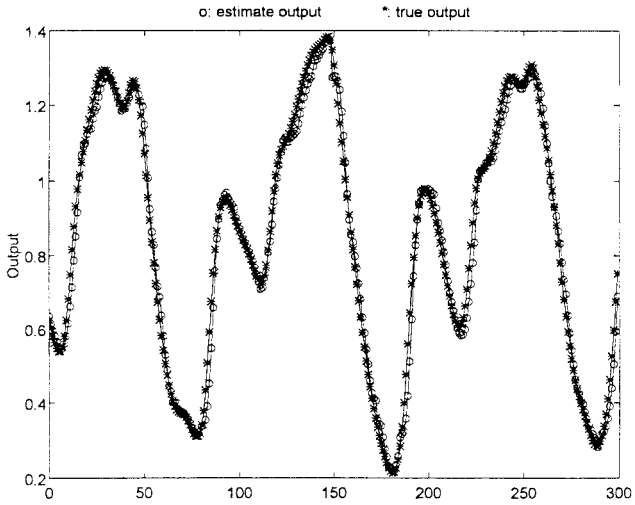


Fig. 23. Simulation results of the time-series prediction from $x(701)$–$x(1000)$ using the FALCON-ART model with 30 fuzzy rules when 700 training data [from $x(1)$–$x(700)$] are used.

learning procedure, the proposed ANFIS can construct an input/output mapping based on both human knowledge (in the form of fuzzy IF–THEN rule) and stipulated input/output data pairs. The ANFIS also has perfect prediction capability on the chaotic time series prediction problem after learning. However, a set of correct fuzzy logic rules and proper input/output space partition must be given in advance by experts before initiating the training of the ANFIS.

Example 3—Control for Backing Up the Truck: Backing a truck to a loading dock is a difficult exercise. It is a nonlinear control problem for which no traditional control design methods exits. Nguyen and Widrow [26] developed a neural network controller which only used numerical data for the backing up the truck problem. Kong and Kosko [27] proposed a fuzzy controller which only used linguistic rules for the same problem. In this example, we develop a controller (called FALCON-ART controller) to back up a simulated truck to a loading dock in a planar parking lot. We use on-line structure-parameter learning algorithm for the FALCON to adaptively generate fuzzy rules and adjust parameters according to training data. This FALCON-ART controller enables the truck to reach the desired position successfully.

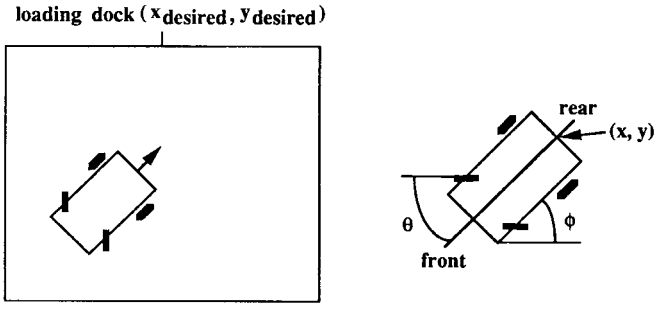


Fig. 24. Diagram of the simulated truck and loading zone.

The simulated truck and loading zone are shown in Fig. 24. The truck position is exactly determined by three state variables ϕ, x, and y, where ϕ is the angle of truck with the horizontal and the coordinate pair (x, y) specifies the position of the rear center of the truck in the plane. The steering angle θ of the truck is the controlled variable. The positive values of θ represent clockwise rotations of the steering wheel and negative values represent counterclockwise rotations. The truck is placed at some initial position and is backed up while being steered by the controller. The objective of this control problem is to use backward movements of the truck only to make the truck arrive at the loading dock at right angle $(\phi_{\text{desired}} = 90°)$ and to have the position of the truck with the desired loading dock $(x_{\text{desired}}, y_{\text{desired}})$. The truck moves backward by fixed distance (d) of the movement of the steering wheel at every step. The loading region is limited to the plane [0, 100] $\times$ [0, 100].

The input and output variables of the FALCON-ART controller must be specified. The controller has two inputs, truck angle ϕ and the cross position x. Assuming enough clearance between the truck and the loading dock, the y coordinate is not considered as an input variable. The output of controller is the steering angle θ. The ranges of the variables x, ϕ and θ are as follows:

$$0 \le x \le 100 \tag{47}$$

$$-90° \le \phi \le 270° \tag{48}$$

$$-30° \le \theta \le 30°. \tag{49}$$

The equations of backward motion of the truck are given by

$$\begin{aligned} x(k+1) &= x(k) + d\cos\theta(k)\cos\phi(k) \\ y(k+1) &= y(k) + d\cos\theta(k)\sin\phi(k) \\ \phi(k+1) &= \tan^{-1}\left[\frac{l\sin\phi(k) + d\cos\phi(k)\sin\theta(k)}{l\cos\phi(k) - d\sin\phi(k)\sin\theta(k)}\right] \end{aligned} \tag{50}$$

where l is the length of the truck. Equation (50) is used to obtain the next state when the present state is given.

For the purpose of training the FALCON-ART controller, learning takes place during several different tries, each starting from an initial state and terminating when the desired state

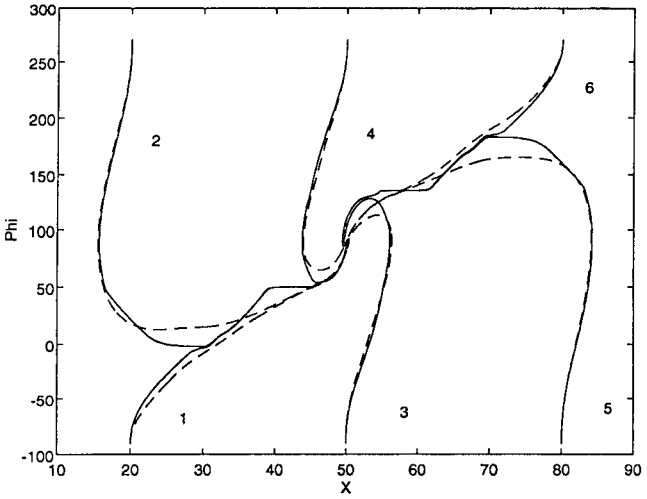


Fig. 25. The moving trajectories of the truck where the solid curves represent the six sets of training trajectories and the dashed curves represent the moving trajectories of the truck under the control of the learned FALCON-ART controller.

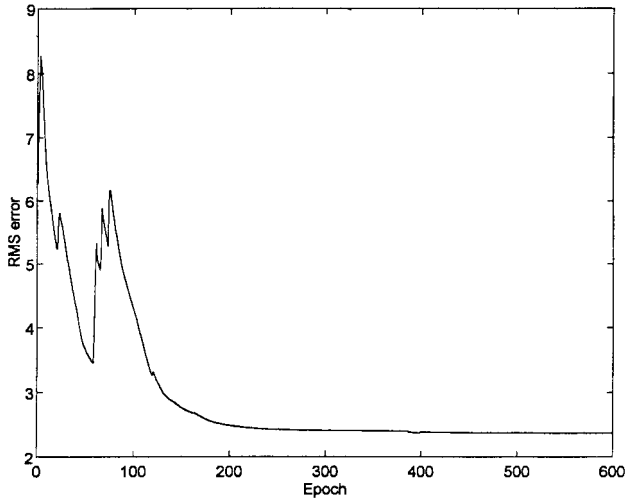


Fig. 26. Learning curve of the FALCON-ART controller in Example 3.

is reached. In our simulation, six different initial positions of the truck are chosen. The six training paths are shown in Fig. 25. The truck moves by a small fixed distance $d = 1.6$ at every step and the length of the truck is set to be $l = 1$. The learning rate $\eta = 0.01$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.6$, $\rho_{\text{output}} = 0.7$ are chosen. The vigilance values decrease gradually with training epoch number increasing. The training process is continued for 600 epochs. In each epoch, all the six sets of training trajectories are presented once to the FALCON-ART controller in a random order. Fig. 26 shows the learning curves for the FALCON-ART controller. After the on-line structure-parameter learning, there are 19 fuzzy logic rules generated in our simulation. The rms error of the controller with nineteen rules approximates to 2.3°. The training result is shown in Fig. 25. In the figure, the solid curves are the training paths and the dotted curves are the paths that the truck runs under the control of the learned controller. As this figure shows, the FALCON-ART controller can smooth the

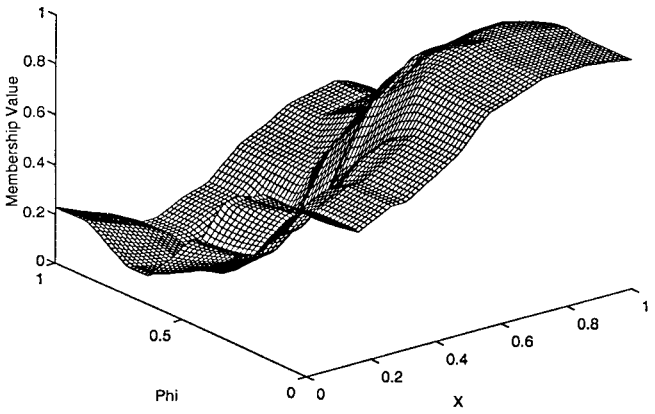


Fig. 27. Three-dimensional (3-D) control surface of the learned FALCON-ART controller in Example 3.

training paths. The three-dimensional (3-D) control surface of the learned FALCON-ART controller in Fig. 27 shows the steering signal outputs with respect to all the combinations of the two input variable values x and ϕ after learning the six sets of training trajectories. After the training process is terminated, Fig. 28(a)–(c) shows the trajectories of the moving truck controlled by the FALCON-ART controller starting from the initial positions $(x, y, \phi) =$ (a) $(40, 20, -30°)$, (b) $(10, 20, -30°)$, and (c) $(30, 20, -250°)$. If we remove the truck training trajectories 3 and 4 from the six sets of training trajectories, we find that the learned FALCON-ART controller can still move the truck nearly to the correct parking position starting at the same initial positions. The trajectory starting at $(x, y, \phi) = (10, 20, -30°)$ in this case is shown in Fig. 29. Although we reduce the number of training patterns, this controller can still move the truck to the correct parking position. This indicates that the FALCON-ART controller can also produce appropriate control action even if training patterns are not distributed over a sufficiently varied area in the state space. Therefore, the FALCON-ART has good generalization capability and robustness.

In the case that six sets of training patterns are used during learning process, we also tried the rule-annihilation technique to combine some similar rules. With this learning, there are twelve fuzzy rules generated in the use of Method 1 of rule annihilation and seven fuzzy logic rules generated in the use of Method 2. The results show that by incorporating the rule-annihilation process into the FALCON-ART, we can obtain fewer fuzzy logic rules that can still move the truck to the correct parking position.

Example 4—Control of the Ball and Beam System: The ball and beam system is shown in Fig. 30. The beam is made to rotate in a vertical plane by applying a torque at the center of rotation; the ball is free to roll along the beam. We require that the ball remain in contact with the beam. The system can be written in the following state-space form:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_2 \\ B(x_1 x_4^2 - G \sin x_3) \\ x_4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u \tag{51}$$

$$y = x_1 \tag{52}$$

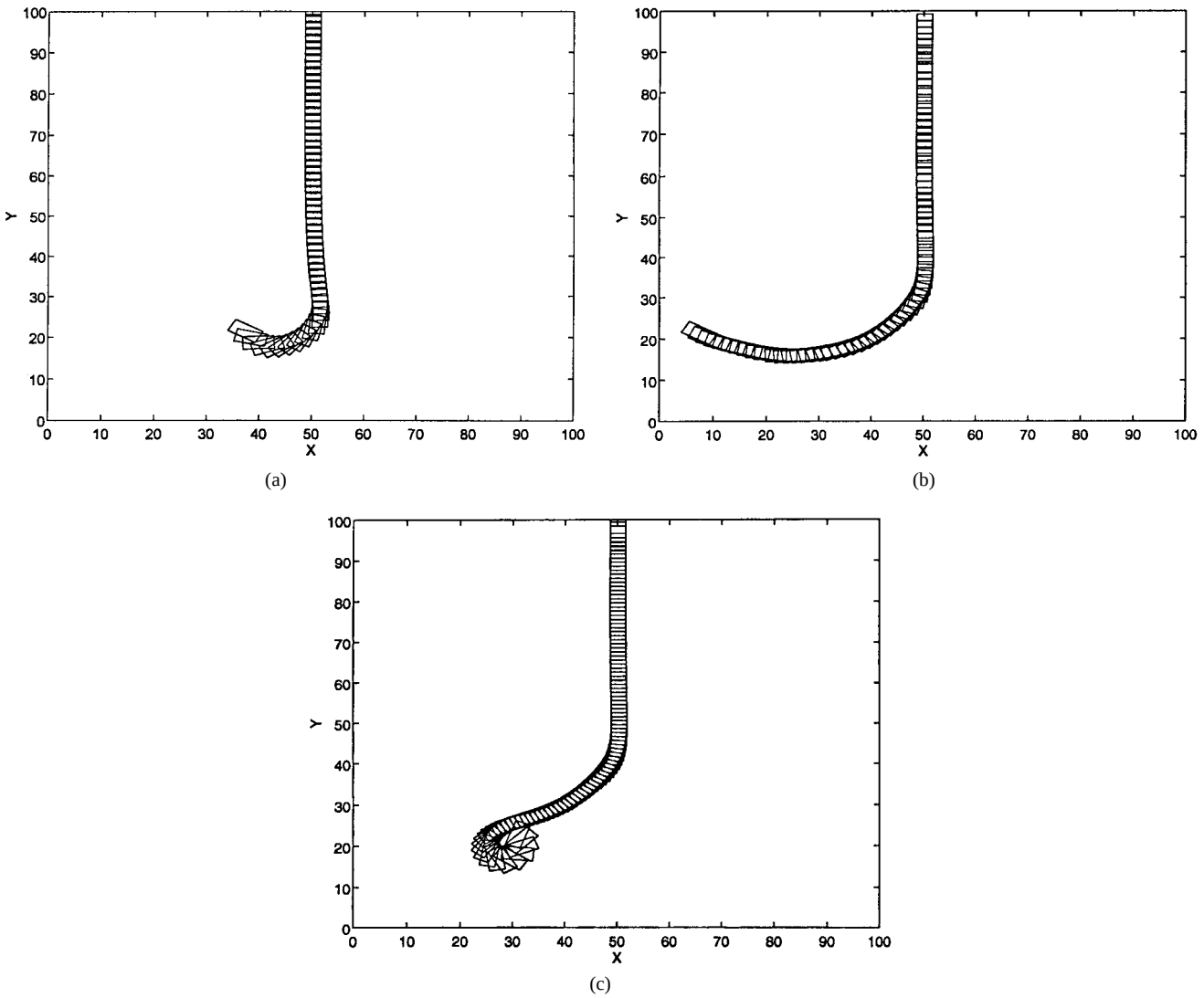


Fig. 28. Truck moving trajectories starting at three different initial positions under the control of the FALCON-ART model after learning six sets of training trajectories.

where $\mathbf{x} = (x_1, x_2, x_3, x_4)^T \equiv (r, \dot{r}, \theta, \dot{\theta})^T$ is the state of the system and $y = x_1 \equiv r$ is the output of the system. The control u is the angular acceleration $(\ddot{\theta})$ and the parameters $B = 0.7143$ and $G = 9.81$ are chosen in this system. The purpose of control is to determine $u(\mathbf{x})$ such that the closed-loop system output y will converge to zero from different initial conditions.

According to the input/output-linearization algorithm [28], the control law $u(\mathbf{x})$ is determined as follows: for state $\mathbf{x}$, compute $v(\mathbf{x}) = -\alpha_3\phi_4(\mathbf{x}) - \alpha_2\phi_3(\mathbf{x}) - \alpha_1\phi_2(\mathbf{x}) - \alpha_o\phi_1(\mathbf{x})$, where $\phi_1(\mathbf{x}) = x1$, $\phi_2(\mathbf{x}) = x2$, $\phi_3(\mathbf{x}) = -BG \sin x_3$, $\phi_4(\mathbf{x}) = -BGx_4 \cos x_3$, and the α_i are chosen so that $s^4 + \alpha_3 s^3 + \alpha_2 s^2 + \alpha_1 s + \alpha_o$ is a Hurwitz polynomial. Compute $a(\mathbf{x}) = -BG \cos x_3$ and $b(\mathbf{x}) = BGx_4^2 \sin x_3$; then $u(\mathbf{x}) = [v(\mathbf{x}) - b(\mathbf{x})]/a(\mathbf{x})$.

In our simulation, we solve the differential equations using the second/third-order Runge–Kutta method. We train the FALCON-ART model to approximate the aforementioned conventional controller of a ball and beam system. Let all closed-loop poles be placed at -2. Thus, $\alpha_o = 16$, $\alpha_1 = 32$, $\alpha_2 = 24$, and $\alpha_3 = 8$ are chosen. We use $u(\mathbf{x}) = [v(\mathbf{x}) - b(\mathbf{x})]/a(\mathbf{x})$ to generate the input/output

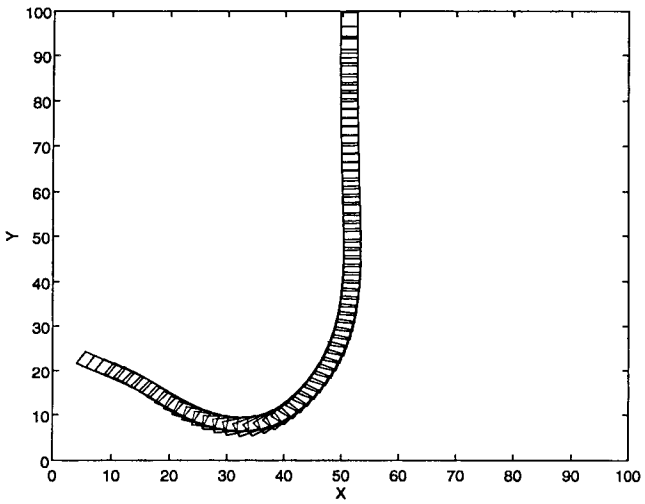


Fig. 29. Truck moving trajectory under the control the FALCON-ART model after learning four sets of training trajectories.

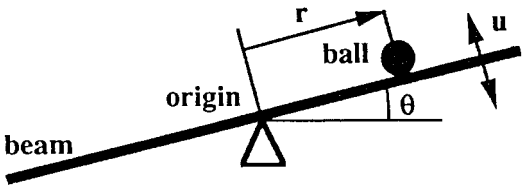


Fig. 30. The ball and beam system.

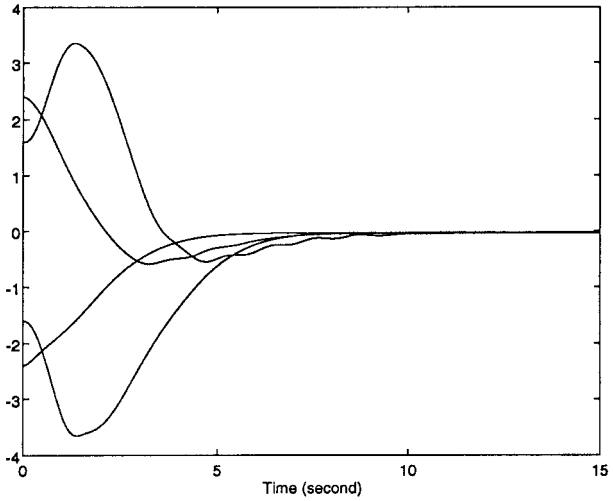


Fig. 31. The responses of the closed-loop ball and beam system controlled by the FALCON-ART model for four initial conditions.

training pair with $\mathbf{x}$ obtained from randomly sampling 300 points in the region $U = [-4, 4] \times [-3, 3] \times [-1, 1] \times [-2, 2]$. The learning rate $\eta = 0.001$, sensitivity parameter $\gamma = 4$, and vigilance parameter $\rho_{\text{input}} = 0.3$, $\rho_{\text{output}} = 0.7$ are chosen. After the on-line structure-parameter learning, there are 28 fuzzy logic rules generated in our simulation. We have tested the learned controller at four initial conditions: $\mathbf{x}(0) = [2.4, -0.1, 0.6, 0.1]^T$, $[1.6, 0.05, -0.5, -0.05]^T$, $[-1.6, -0.05, 0.5, 0.05]^T$, and $[-2.4, 0.1, -0.6, -0.1]^T$. Fig. 31 shows the output responses of the closed-loop ball and beam system controlled by the FALCON-ART model. These responses approximate those of the original controller for the four initial conditions. As a comparison, Fig. 32 shows the output responses of the closed-loop ball and beam system controlled by the input/output-linearization algorithm for four initial conditions. We also show the behavior of the four states of the ball and beam system starting at the initial condition $[-2.4, 0.1, -0.6, -0.1]^T$ in Fig. 33. In this figure, the four states of the system decay to zero gradually. The results show the perfect control capability of the trained FALCON-ART model.

VI. Discussion

In this section, we summarize the features of the proposed FALCON-ART model. First, distributed representation is used to represent the input patterns in the FALCON-ART model. This is achieved by the fuzzification process through the adaptive input membership functions. With the adaptive input membership functions, the input space is divided into overlapping smaller regions and, more importantly, this partitioning is

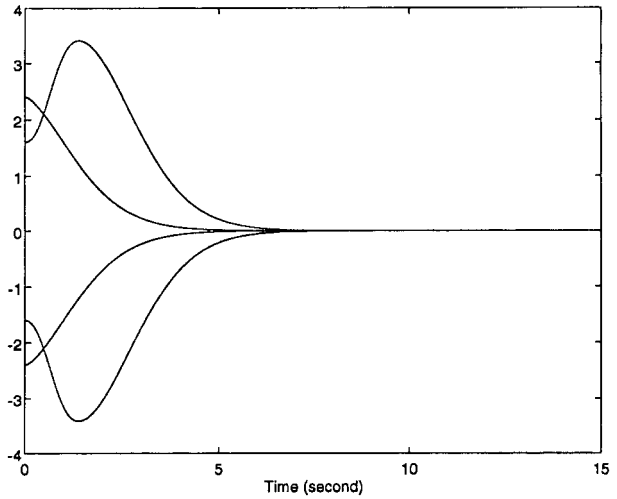


Fig. 32. The responses of the closed-loop ball and beam system controlled by the input/output-linearization algorithm for four initial conditions.

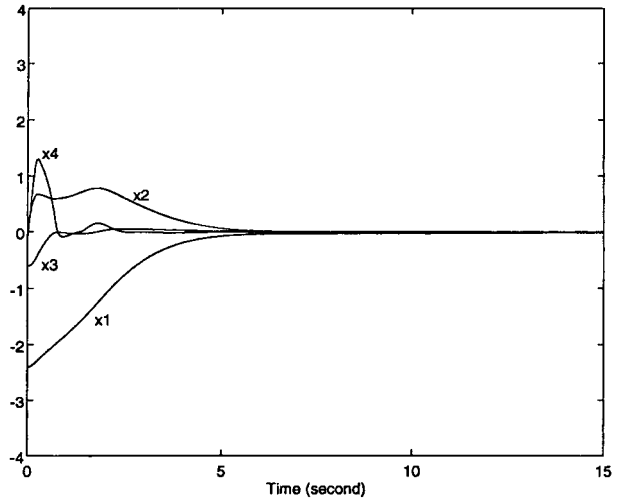


Fig. 33. The responses of the four states of the ball and beam system under the control of the learned FALCON-ART controller.

not performed in advance, but is dynamically and appropriately adjusted during the learning process. As a result, each region varies in size and the degree of overlapping between regions is also adjustable. This is in contrast to the Wang and Mendel's system [17], ANFIS system [9], and Kosko's system [14] in which the input space need to be divided properly in advance. The second feature of the proposed FALCON-ART model is its dynamic structure-parameter-learning ability that can find proper fuzzy logic rules. This is in contrast to the approaches in [9] and [18], which need *a priori* control knowledge from expert operators in term of fuzzy control rules. The third feature of the proposed FALCON-ART model is its rule-annihilation ability. The FALCON-ART model has the ability to delete unnecessary or redundant rules. In other words, it can combine some nodes with the same or a very similar control action. The fourth feature of the proposed FALCON-ART model is the ease with which expert knowledge can be incorporated into the network greatly shortening learning time. The fifth feature of the proposed FALCON-ART model is that

it flexibly partitions the input/output spaces according to the distribution of environment states. This avoids the combinatorial growth problem encountered by partitioned grids in some complex systems. In addition to the simulations done in this paper, the proposed supervised structure-parameter-learning algorithm has been used to solve many practical problems, including crane-position control and adaptive control of electrodischarge machining (EDM) in our laboratory.

VII. Conclusion

In this paper, we introduced a general connectionist model of a fuzzy logic control system called FALCON. An on-line structure-parameter learning algorithm called FALCON-ART was proposed for constructing the FALCON dynamically. The proposed learning algorithms is able to partition the input and output spaces and then find proper fuzzy rules and optimal membership functions dynamically. The FALCON-ART partitions the pattern space into irregular hyperboxes and, thus, can avoid the problem of combinatorial growing of partitioned grids in some complex systems. Simulations demonstrate that the proposed FALCON-ART model is quite effective in many applications.

References

[1] R. R. Yager, "Implementing fuzzy logic controller using a neural network," *Fuzzy Sets Syst.*, vol. 48, pp. 53–64, 1992.

[2] P. J. Werbos, "Neural control and fuzzy logic: Connections and designs," *Int. J. Approx. Reasoning*, vol. 6, pp. 185–219, 1992.

[3] T. Tagaki and I. Hayashi, "NN-driven fuzzy reasoning," *Int. J. .Approx. Reasoning*, vol. 5, pp. 191–212, 1991.

[4] H. R. Berenji, "An architecture for designing fuzzy controllers using neural networks," *Int. J. Approx. Reasoning*, vol. 6, pp. 267–292, 1991.

[5] H. Nomura, I. Hayashi, and N. Wakami, "A learning method of fuzzy inference rules descent method," in *Proc. IEEE Int. Conf. Neural Networks*, Baltimore, MD, June 1992, pp. 203–210.

[6] D. Nauck and R. Kruse, "A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backpropagation," in *Proc. IEEE Int. Conf. Neural Networks*, San Francisco, CA, Mar. 1993, pp. 1022–1027.

[7] C. T. Sun and J. S. Jang, "A neuro-fuzzy classifier and its applications," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Francisco, CA, Mar. 1993, pp. 94–98.

[8] J. S. Jang, "Self-learning fuzzy controllers based on temporal back propagation," *IEEE Trans. Neural Networks*, vol. 3, pp. 723–741, May 1992.

[9] ——, "ANFIS: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 3, pp. 665–685, May/June 1993.

[10] H. Bersini, J. P. Nordvik, and A. Bonarini, "A simple direct adaptive fuzzy controller derived from its neural equivalent," in *Proc. IEEE Int. Conf. Neural Networks*, San Francisco, CA, Mar. 1993, pp. 345–350.

[11] I. Enbutsu, K. Baba, and N. Hara, "Fuzzy rule extraction from a multilayered neural network," in *Proc. IEEE Int. Joint Conf. Neural Networks*, Seattle, WA, July 1991, pp. 461–465.

[12] C. C. Lee and H. R. Berenji, "An intelligent controller based on approximate reasoning and reinforcement learning," in *Proc. IEEE Intell. Mach.*, Albany, NY, Sept. 1989, pp. 200–205.

[13] E. Khan and P. Venkatapuram, "Neufuz: Neural network based fuzzy logic design algorithms," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Francisco, CA, Mar. 1993, pp. 647–654.

[14] B. Kosko, *Neural Networks and Fuzzy Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1992.

[15] L. X. Wang and J. M. Mendel, "Backpropagation fuzzy rules as nonlinear dynamic system identifiers," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Diego, CA, Mar. 1992, pp. 1409–1418.

[16] ——, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks*, vol. 3, pp. 807–814, May 1992.

[17] ——, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 6, pp. 1414–1427, Nov./Dec. 1992.

[18] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, pp. 724–740, May 1992.

[19] C. T. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, pp. 1320–1336, Dec. 1991.

[20] ——, "Real-time supervised structure/parameter learning for fuzzy neural network," in *IEEE Int. Conf. Fuzzy Syst.*, San Diego, CA, Mar. 1992, pp. 1283–1290.

[21] ——, "Reinforcement structure/parameter learning for an integrated fuzzy neural network," *IEEE Trans. Fuzzy Syst.*, vol. 2, pp. 46–63, Feb. 1994.

[22] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.

[23] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. Neural Networks*, vol. 3, pp. 698–712, May 1992.

[24] P. K. Simpson, "Fuzzy min-max neural networks—Part 2: Clustering," *IEEE Trans. Fuzzy Syst.*, vol. 1, pp. 32–45, Feb. 1993.

[25] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–26, Jan. 1990.

[26] D. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural network," *IEEE Contr. Syst. Mag.*, vol. 10, no. 3, pp. 18–23, Apr. 1990.

[27] S. G. Kong and B. Kosko, "Comparison of fuzzy and neural truck backer-upper control systems," in *Int. Joint Conf. Neural Network*, San Diego, CA, June 1990, vol. 3, pp. 349–358.

[28] J. Hauser, S. Sastry, and P. Kokotovic, "Nonlinear control via approximate input–output linearization: The ball and beam example," *IEEE Trans. Automat. Contr.*, vol. 37, no. 3, pp. 392–398, Mar. 1992.

Cheng-Jian Lin (S'94–M'95) received the B.S. degree in electrical engineering from Tatung Institute of Technology, Taiwan, R.O.C., in 1986, and the M.S. and Ph.D. degrees in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1991 and 1996, respectively.

Currently, he is an Associate Professor in the Department of Electronic Engineering of Nankai College of Technology and Commerce, Nantou, R.O.C. His current research interests are neural networks, learning systems, fuzzy control, approximate reasoning, signal processing, and image processing.

Dr. Lin is a member of the IEEE Control Systems Society and the IEEE Systems, Man, and Cybernetics Society.

Chin-Teng Lin received the B.S. degree in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1986, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1989 and 1992, respectively.

Since August 1992, he has been with the College of Electrical Engineering and Computer Science, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., where he is currently a Professor of control engineering. He is the co-author of *Neural Fuzzy Systems—A Neuro-Fuzzy Synergism to Intelligent Systems* (Englewwod Cliffs, NJ: Prentice-Hall, 1996) and the author of *Neural Fuzzy Control Systems with Structure and Parameter Learning* (River Edge, NJ: World Scientific, 1994). His current research interests are fuzzy systems, neural networks, intelligent control, human-machine interface, and video and audio processing.

Dr. Lin is a member of Tau Beta Pi and Eta Kappa Nu. He is also a member of the IEEE Computer Society, the IEEE Robotics and Automation Society, and the IEEE Systems, Man, and Cybernetics Society.