ELSEVIER

Fuzzy Sets and Systems 112 (2000) 65–84

FUZZY sets and systems

www.elsevier.com/locate/fss

A GA-based fuzzy adaptive learning control network

I-Fang Chung, Cheng-Jian Lin*, Chin-Teng Lin

Department of Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, ROC

Received May 1997; received in revised form March 1998

Abstract

This paper addresses the structure and an associated 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. A structure/parameter learning algorithm, called FALCON-GA, is proposed for constructing the FALCON automatically. The FALCON-GA is a three-phase hybrid learning algorithm. In the first phase, the fuzzy ART algorithm is used to do fuzzy clustering in the input/output spaces according to the supervised training data. In the second phase, the genetic algorithm (GA) is used to find proper fuzzy logic rules by associating input clusters and output clusters. Finally, in the third phase, the backpropagation algorithm is used for tuning input/output membership functions. Hence, the FALCON-GA combines the backpropagation algorithm for parameter learning and both the fuzzy ART and GAs for structure learning. It can partition the input/output spaces, tune membership functions and find proper fuzzy logic rules automatically. The proposed FALCON has two important features. First, it reduces the combinatorial demands placed by the standard methods for adaptive linearization of a system. Second, the FALCON is a highly autonomous system. In its learning scheme, 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. Computer simulations have been conducted to illustrate the performance and applicability of the proposed system. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Fuzzy ART; Expert systems; Membership function; Space partition; Backpropagation; Chaotic sequence

1. Introduction

Bringing the learning abilities of neural networks to automate and realize the design of fuzzy control systems has become an active research area in recent years [2,10,14,17,18,25–27,30–32]. Developed techniques include the automatic generation of fuzzy rules from numerical data [14,18,25,27,30] and the tuning of membership functions [2,10,17]. 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. Recently, due to its global optimization capability, the genetic algorithm has become another useful tool to the automatic design of fuzzy control systems [5,8,9,11,12,16,22,28]. Such synergism of GAs, neural networks and fuzzy logic into a functional system provides a new direction toward the realization of intelligent systems for various applications.

A genetic algorithm (GA) is a parallel, global search technique that emulates natural evolution. Because it

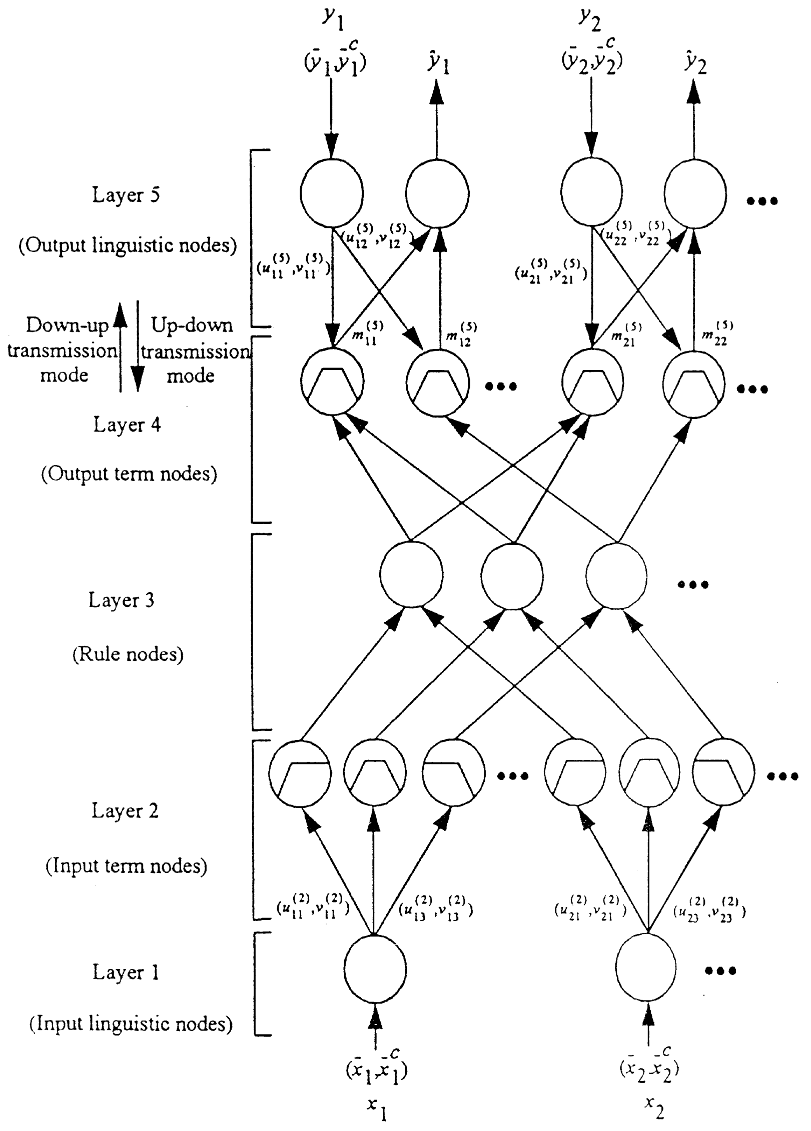
* Corresponding author.

simultaneously evaluates many points in the search space, it is more likely to converge toward the global solution. A GA applies operators inspired by the mechanics of natural selection to a population of binary string encoding the parameter space at each generation, it explores different areas of the parameter space, and then directs the search to regions where there is a high probability of finding improved performance. By working with a population of solutions, the algorithm can in effect seek many local minima and thereby increase the likelihood of finding the global minima.

GAs have been employed for generating fuzzy rules and adjusting membership functions in fuzzy logic systems. The pioneer was Karr [11,12] who used GAs to adjust membership functions. Nomura et al. [22] used GAs to determine the fuzzy partition of input spaces. Hence both the number of fuzzy terms for each input variable and the membership function of each fuzzy term could be determined. In [28], an appropriate fuzzy set in the consequent part of each fuzzy rule was selected using GAs. Lee and Takagi [16] used GAs to approach simultaneous membership function and rule set design. Homaifar [8] also examined the applicability of GAs in the simultaneous design of membership functions and rule sets for fuzzy logic controllers. Ishibuchi [9] proposed a GA-based method for selecting a small number of significant fuzzy rules to construct a compact fuzzy classification system with high classification power. GAs also provide more flexibility in designing the rule set and membership functions for some difficult non-linear control problems [4].

In this paper, we apply the GA to our fuzzy adaptive learning control network proposed previously in [18] to enhance its learning ability. The proposed fuzzy adaptive learning control network (FALCON) can be constructed automatically by learning from training data. 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 network, and the other is for a decision signal (actual output) to be pumped out of the network. Nodes at layers two and four are term nodes that act as membership functions to represent the terms of the respective linguistic variable. Each node at layer three is a rule node that 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 a hybrid learning algorithm, called FALCON-GA. We shall call a FALCON with this learning algorithm the FALCON-GA model. The FALCON-GA model has some important properties as described below. In many existing fuzzy or neuro-fuzzy control systems, the input and output spaces are partitioned into “grids”. When the number of input/output variables increase, the number of 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. To avoid the problem of combinatorial growing of partitioned grids in complex systems, a more flexible and irregular space partitioning method has been developed [18] and is further extended in this paper. The problem of space partitioning from numerical training data is basically a clustering problem. The proposed FALCON-GA applies the Fuzzy Adaptive Resonance Theory (Fuzzy ART) proposed by Carpenter et al. [3] to do fuzzy clustering in the input/output spaces. It then applies the GA to find proper fuzzy logic rules by associating input clusters and output clusters, where an input cluster forms a precondition and an output cluster forms a consequent of a determined fuzzy rule. Finally, the backpropagation algorithm is used to tune the input/output membership functions optimally to meet the desired input/output mapping. Hence, the FALCON-GA is a three-phase hybrid learning algorithm combining both the fuzzy ART (phase 1) and the GA (phase 2) for structure learning, and the backpropagation algorithm (phase 3) for parameter learning. The FALCON-GA can partition the input/output spaces, tune membership functions and find proper fuzzy logic rules automatically. More notably, in this learning scheme, only the training data need to be pro-



y_1
$(\bar{y}_1, \bar{y}_1^c)$
$\hat{y}_1$
y_2
$(\bar{y}_2, \bar{y}_2^c)$
$\hat{y}_2$
Layer 5
(Output linguistic nodes)
Down-up transmission mode
Up-down transmission mode
Layer 4
(Output term nodes)
Layer 3
(Rule nodes)
Layer 2
(Input term nodes)
Layer 1
(Input linguistic nodes)
$(u_{11}^{(5)}, v_{11}^{(5)})$
$(u_{12}^{(5)}, v_{12}^{(5)})$
$(u_{21}^{(5)}, v_{21}^{(5)})$
$(u_{22}^{(5)}, v_{22}^{(5)})$
$m_{11}^{(5)}$
$m_{12}^{(5)}$
$m_{21}^{(5)}$
$m_{22}^{(5)}$
$(u_{11}^{(2)}, v_{11}^{(2)})$
$(u_{13}^{(2)}, v_{13}^{(2)})$
$(u_{21}^{(2)}, v_{21}^{(2)})$
$(u_{23}^{(2)}, v_{23}^{(2)})$
$(\bar{x}_1, \bar{x}_1^c)$
x_1
$(\bar{x}_2, \bar{x}_2^c)$
x_2

Fig. 1. The proposed fuzzy adaptive learning control network (FALCON).

vided from the outside world. The users need not give the initial fuzzy partitions, membership functions and fuzzy logic rules. In other words, there are no input/output term nodes and no rule nodes at the beginning of learning. The whole network structure of the FALCON is created in the learning process.

This paper is organized as follows. A brief introduction to GAs is made in Section 2. Section 3 describes the structure of the FALCON-GA model. The structure/parameter learning algorithm, FALCON-GA, which combines the fuzzy ART algorithm, GA and backpropagation algorithm is presented in Section 4. In Section 5, the FALCON-GA model is used to control the truck backer-upper problem and predict the chaotic time-series to demonstrate its learning capability. Conclusions are summarized in the last section.

2. Hybrid genetic algorithms

The basic element processed by a GA is the string formed by concatenating substrings, each of which is a binary coding of a parameter of the search space. Thus, each string represents a point in the search space and hence a possible solution to the problem. Each string is decoded by an evaluator to obtain its objective function value. This function value, which should be maximized or minimized by the GA, is then converted to a fitness value that determines the probability of the individual undergoing genetic operators. The population then evolves from generation to generation through the application of the genetic operators. The total number of strings included in a population is kept unchanged through generations. A GA in its simplest form uses three operators: *reproduction*, *crossover* and *mutation* [5]. Through reproduction, strings with high fitnesses receive multiple copies in the next generation while strings with low fitnesses receive fewer copies or even none at all. The crossover operator produces two offspring (new candidate solutions) by recombining the information from two parents in two steps. First, a given number of crossing sites are selected along the parent strings uniformly at random. Second, two new strings are formed by exchanging alternate pairs of selection between the selected sites. In the simplest form, crossover with a single crossing site refers to taking a string, splitting it into two parts at a randomly generated crossover point and recombining it with another string that has also been split at the same crossover point. This procedure serves to promote a change in the best strings which could give them even higher fitnesses. Mutation is the random alteration of a bit in the string which assists in keeping diversity in the population.

Traditional simple GAs, though robust, are generally not the most successful optimization algorithm on any particular domain. Hybridizing a GA with algorithms currently in use can produce an algorithm better than the GA and the current algorithms. Hence, for an optimization problem, when there exist algorithms, optimization heuristics, or domain knowledge that can aid in optimization, it may be advantageous to consider a hybrid GA. GAs may be crossed with various problem-specific search techniques to form a hybrid that exploits the global perspective of the GA (*global search*) and the convergence of the problem-specific technique (*local search*).

There are numerous gradient techniques (e.g., gradient descent method, conjugate gradient method) and gradientless techniques (e.g., golden search, simplex method) available for finding local optimal in a calculus-friendly function (e.g., continuous function) [19]. Even without a calculus-friendly function, there are well developed heuristic search schemes for many popular problems. For example, the greedy algorithms in combinatorial optimization are a form of local search [15]. An intuitive concept of hybridizing GAs with these local search techniques is that the GA finds the hills and the local searcher goes and climbs them. Thus, in this approach, we simply allow the GA to run to substantial convergence and then we permit the local optimization procedure to take over, perhaps searching from the top 5% or 10% of points in the last generation.

In some situations, hybridization entails using the representation as well as optimization techniques already in use in the domain, while tailoring the GA operators to the new representation. Moreover, hybridization can entail adding domain-based optimization heuristics to the GA operator set. In these cases, we can no longer apply the familiar GA operators directly and must create their analogs to account for new representations and/or added optimization schemes. For example, Davis [4] and Adler [1] describe an approach to hybridizing a simple GA with the simulated annealing algorithm, and Tsinas [29] and Petridis [23] describe an approach to hybridizing a simple GA with the backpropagation algorithm. A hybrid of both categories of learning methods result in a more powerful, more robust and faster learning procedure. In this paper, we shall hybrid GA with fuzzy ART and backpropagation algorithms to form the FALCON-GA learning algorithm.

3. The structure of the FALCON-GA model

In this section, we describe the structure and functions of the proposed FALCON-GA model. The FALCON (see Fig. 1) has five layers with node and link numbering defined by the brackets on the left-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 [3] to normalize the input/output training vectors. Complement coding is a normalization process that rescales an n-dimensional vector in $\mathbb{R}^n, \boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, to its $2n$-dimensional complement coding form in $[0,1]^{2n}, \boldsymbol{x}'$, such that

$$\begin{aligned} \boldsymbol{x}' &\equiv (\bar{x}_1, \bar{x}_1^{\mathrm{c}}, \bar{x}_2, \bar{x}_2^{\mathrm{c}}, \ldots, \bar{x}_n, \bar{x}_n^{\mathrm{c}}) \\ &= (\bar{x}_1, 1-\bar{x}_1, \bar{x}_2, 1-\bar{x}_2, \ldots, \bar{x}_n, 1-\bar{x}_n), \end{aligned} \tag{1}$$

where $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) = \bar{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|$ and $\bar{x}_1^{\mathrm{c}}$ is the complement of $\bar{x}_1$, i.e. $\bar{x}_1^{\mathrm{c}} = 1 - \bar{x}_1$. As mentioned in [3], complement coding helps avoid the problem of category proliferation. The category proliferation happens when we use fuzzy-ART for fuzzy clustering procedure. In fuzzy clustering, to satisfy the resonance criterion (Eqs. (11)–(13)), future non-complement coding inputs will be forced to drag other weight vectors toward the origin, or to choose uncommitted nodes, even though the choice value of these nodes is small. Therefore, we must use complement coding in input data preprocessing to avoid this problem. Additionally, the complement coding also represents a training vector and achieves normalization while preserving 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 weighted fan-in connections from other nodes, and fan-out connections to other nodes. Associated with the fan-in of a node is an integration function f that combines information, activation, or evidence from other nodes, and provides the net input; i.e.

$$\text{net-input} = f(z_1^{(k)}, z_2^{(k)}, \ldots, z_p^{(k)}; w_1^{(k)}, w_2^{(k)}, \ldots, w_p^{(k)}), \tag{2}$$

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,

$$\text{output} = a(f(\cdot)), \tag{3}$$

where $a(\cdot)$ denotes the activation function. We shall next 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,

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

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, $\bar{x}_i$ and $\bar{x}_i^{\mathrm{c}} = 1 - \bar{x}_i$.

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 membership function. The following trapezoidal membership function [24] is used:

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

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. 2(a)); $z_{ij}^{(2)}$ is the input to the jth input term node from the ith input linguistic node (i.e., $z_{ij}^{(2)} = \bar{x}_i$); and

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

The parameter γ_{ij} is the sensitivity parameter that regulates the fuzziness of the trapezoidal membership function. A large γ_{ij} means a more crisp fuzzy set, and a smaller γ_{ij} 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 Eq. (5). The two weights, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, correspond respectively to the two inputs, $\bar{x}_i$ and $\bar{x}_i^{\mathrm{c}}$ from the input linguistic node i. More precisely, $\bar{x}_i$ and $\bar{x}_i^{\mathrm{c}}$, the two inputs to the input term node j, will be used during the fuzzy-ART clustering process in the first learning phase to decide $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, respectively. In FALCON's other learning phases and normal operating, only $\bar{x}_i$ is used in the forward reasoning process (i.e. $z_{ij}^{(2)} = \bar{x}_i$ in Eq. (5)).

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). Note that each input linguistic variable has the same number of terms in the FALCON, as mentioned above. The links in layer 3 are used to perform precondition matching of fuzzy logic rules. Hence the rule nodes perform the product operation,

$$f(z_i^{(3)}) = \prod_{i=1}^{n} z_i^{(3)} \quad \text{and} \quad a(f(\cdot)) = f(\cdot), \tag{7}$$

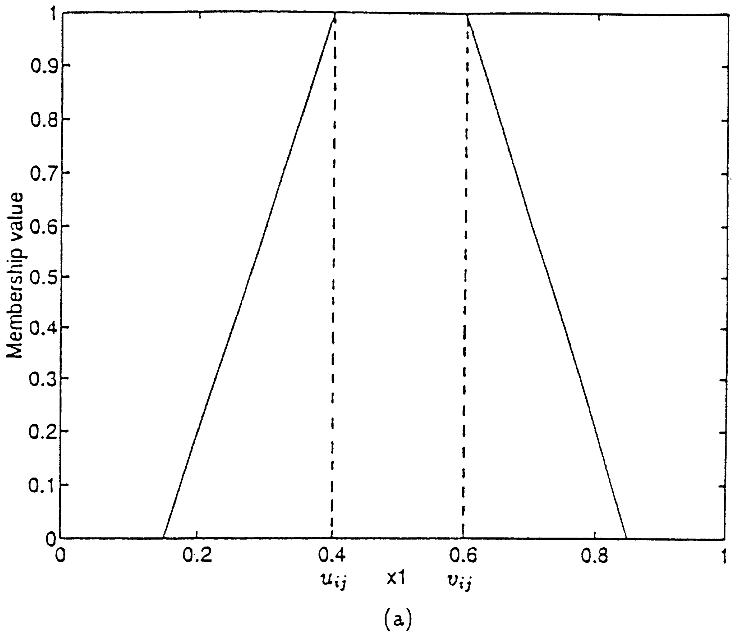
where $z_i^{(3)}$ is the ith input to a node in layer 3 and the product is over the inputs of this node. The link weight in layer 3 ($w_i^{(3)}$) is then unity. Note that the product operation in the above equation is equivalent to defining a multidimensional (n-dimensional) membership function, which is the product of the trapezoid functions in Eq. (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, $[(u_{1j}^{(2)}, v_{1j}^{(2)}), \ldots, (u_{ij}^{(2)}, v_{ij}^{(2)}), \ldots, (u_{nj}^{(2)}, v_{nj}^{(2)})]$, defines a hyperbox in the input space. An illustration of a two-dimensional hyperbox membership function is shown in Fig. 2(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. We shall use the GA to determine the connection types of the links at layers 3 and 4. Note that different rule nodes may be connected to the same output hyperbox (i.e. they may have the same consequent) as 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,

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

and

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



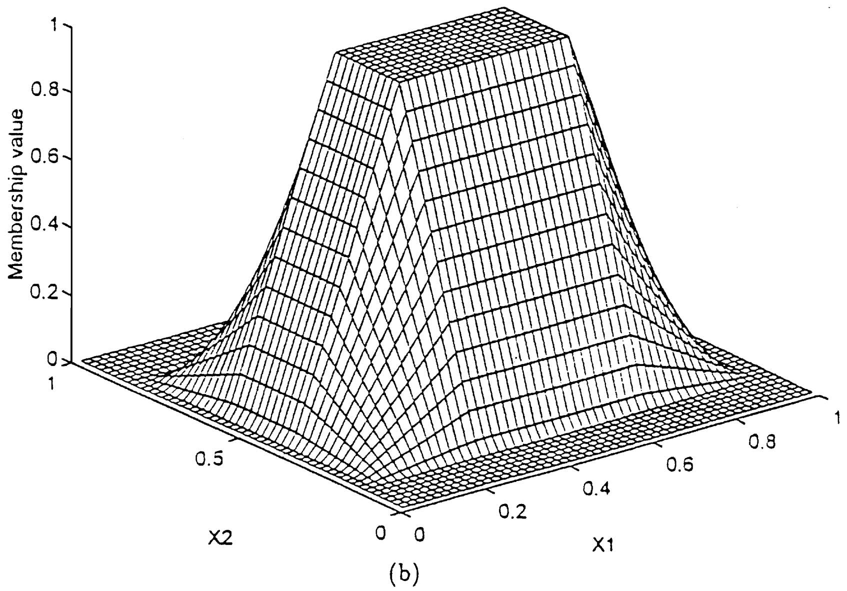


Fig. 2. (a) One-dimensional trapezoidal membership function. (b) Two-dimensional trapezoidal membership function.

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 is $w_i^{(4)} = 1$. 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 one-dimensional 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 $[(u_{1j}^{(5)}, v_{1j}^{(5)}), \ldots, (u_{ij}^{(5)}, v_{ij}^{(5)}), \ldots, (u_{mj}^{(5)}, v_{mj}^{(5)})]$, defines a hyperbox in the output space.

Layer 5: Each node in this layer is called an 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

$$f(\bar{y}_i, \bar{y}_i^{\mathrm{c}}) = (\bar{y}_i, \bar{y}_i^{\mathrm{c}}) = (\bar{y}_i, 1 - \bar{y}_i)$$

and

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

where $\bar{y}_i$ is the ith element of the normalized desired output vector. Note that complement coding is also performed on the desired output vectors. Hence, 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:

$$f(z_j^{(5)}) = \sum_j w_{ij}^{(5)} z_j^{(5)} = \sum_j m_{ij}^{(5)} z_j^{(5)}$$

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.

Based on the above structure, a learning algorithm, FALCON-GA, will be proposed in the next section to determine the proper corners of the hyperbox (u_{ij}'s and v_{ij}'s) for each term node in layers 2 and 4. Also, it will learn fuzzy logic rules and connection types of the links at layers 3 and 4; that is, the precondition links and consequent links of the rule nodes.

4. Three-phase hybrid learning algorithm

In this section, we shall present a three-phase hybrid learning scheme for the proposed FALCON-GA model. Fig. 3 shows the flowchart of this learning scheme. This learning scheme, called FALCON-GA, combines fuzzy ART, GA and backpropagation algorithm to partition the input/output spaces, find proper fuzzy logic rules, and adjust membership functions of the FALCON. In phase one (the *fuzzy ART learning phase*), the fuzzy ART algorithm is used to partition the input (output) space by clustering the input (output) training data. In other words, this learning phase is to locate initial membership functions in the input/output spaces. In phase two (the *GA learning phase*), the GA is used to find proper fuzzy rules by associating the input and output membership functions properly. In phase three (the *supervised learning phase*), the backpropagation algorithm is used to adjust the membership functions to minimize the network output error. The FALCON-GA uses both the fuzzy ART and the GA to perform structure

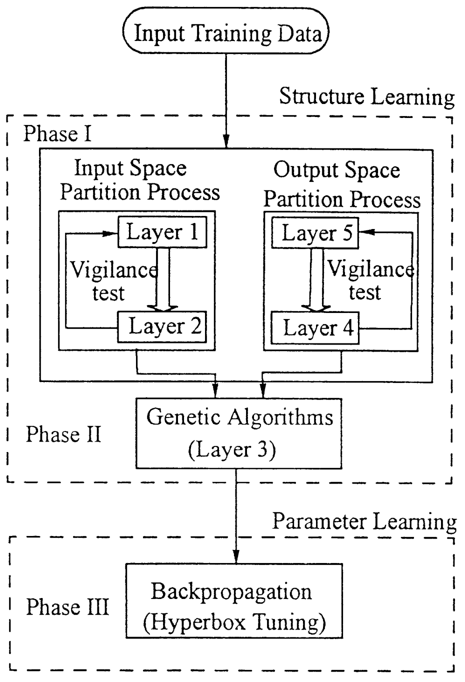


Fig. 3. The flowchart of the learning algorithm for the FALCON-GA model.

learning and the backpropagation algorithm to perform parameter learning. With the proposed learning scheme, 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 initial structure of the FALCON. They are all created during the learning process.

4.1. The fuzzy ART learning phase

The problem for the fuzzy ART learning can be stated as: given a set of input training data, $x_i(t)$, $i=1,\dots,n$ and desired output value $y_i(t)$, $i=1,\dots,m$, $t=1,2,\dots$, we try to find proper fuzzy partitions and membership functions in the input/output spaces. In this phase, the network works in a two-sided manner, that is, the nodes in layer 4 are in the up–down transmission mode so training input and output data are fed into the network from both sides.

The fuzzy ART learning consists of two learning processes: input fuzzy clustering process and output fuzzy clustering process. The two learning processes are performed simultaneously on both sides of the network, and are described next.

4.1.1. Input fuzzy clustering process

We use the fuzzy ART fast learning algorithm [3] 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 component coded input vector $\boldsymbol{x}'$ (see Eq. (1)), the values of choice functions, T_j, are computed by

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

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

$$T_J=\max\{T_j\colon j=1,\dots,N\}. \tag{12}$$

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

$$\frac{|\boldsymbol{x}'\wedge \boldsymbol{w}_J|}{|\boldsymbol{x}'|}\geqslant\rho, \tag{13}$$

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 0 for the duration of the input presentation to prevent the persistent selection of the same category during a search (we call this action "disabling J"). A new index J is then chosen using Eq. (12). The search process continues until the chosen J satisfies Eq. (13). This search process is indicated by the feedback arrow marked with "vigilance test" in Fig. 3. 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, $\boldsymbol{x}'$. These newly added input term nodes and links define a new hyperbox, and thus a new category, in the input space.

4.1.2. *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, $\boldsymbol{y}' = (\bar{y}, \bar{y}^{c}) = (\bar{y}, 1 - \bar{y})$.

4.2. *The genetic algorithm learning phase*

After the initial membership functions are located by performing fuzzy clustering in the input/output spaces in the first learning phase, the network then enters the second learning phase to find proper fuzzy logic rules by associating input clusters and output clusters. The GA is used to find fuzzy rules by determining the links between layers 3 and 4 in this phase. The following steps are employed to generate and handle a set of strings (i.e. a population) in the second learning phase.

Step 1. *Initialization*: The first step in GAs is coding, which maps a finite-length string to the searched parameters. We first generate an initial population containing N_{pop} strings, where N_{pop} is the number of strings in each population. Each string is an individual point in the search space. The binary coding scheme is used in the FALCON-GA model. This means that each string is generated by randomly assigning a binary value, $\{0, 1\}$, to each possible fuzzy rule between layers 3 and 4, value 1 indicating the existence of fuzzy rule, whereas value 0 indicating a dummy rule. Each string thus represents a set of possible fuzzy rules. From the whole rule space, a population is chosen.

Step 2. *Fitness function*: In this step, each string is decoded by an evaluator into an objective function value. This function value, which should be maximized by the GA, is then converted to a fitness value, $FIT(i)$, where $FIT(\cdot)$ is a fitness function. A fitness value is assigned to each individual in the population, where high values mean a good fit. The fitness function can be any non-linear, non-differentiable, or discontinuous positive function, because the GA only needs a fitness value assigned to each string. In the FALCON-GA, the fitness function is defined by

$$FIT(i) = \frac{1}{RMS\_ERROR(i)}, \tag{14}$$

where $RMS\_ERROR(i)$ represents the root-mean-square error between the network outputs and the desired outputs for the ith string (the ith set of fuzzy rules). The goal of the GA learning phase is to maximize the above fitness function.

Step 3. *Reproduction*: Reproduction is a process in which individual strings are copied according to their fitness values, i.e. based on the principle of survival of the fittest. This operator is an artificial version of natural selection. Through reproduction, strings with high fitnesses receive multiple copies in the next generation while strings with low fitnesses receive fewer copies or even none at all.

Step 4. *Crossover*: Reproduction directs the search towards the best existing individuals but does not create any new individuals. In nature, an offspring is rarely an exact clone of a parent. It usually has two parents and inherits genes from both. The main operator in GAs to work on the parents is crossover, which happens for the selected pair with a crossover probability. In the FALCON-GA, we use the two-point crossover operator. At first, two strings from the reproduced population are mated at random, and two crossover points (two bits position) are randomly selected. Then the strings are crossed and separated at these points. For example, the strings before and after the crossover operation could be

$$\begin{array}{ccc} 101|011|10 & & 101|000|10 \\ & \rightarrow & \\ 111|000|11 & & 111|011|11. \end{array}$$

This process makes two new strings, each of which takes after both parents.

Step 5. *Mutation*: Even though reproduction and crossover come up with many new strings, they do not introduce any new information into the population at the bit level. Mutation is the random alteration of bits in the string which assists in keeping diversity in the population. As a source of new bits, mutation is introduced and is applied with a low probability p_{m}. The mutation process is held to escape the local minima in the search space.

Step 6. If the stopping condition is not satisfied (i.e. the current smallest root-mean-square error is not small enough), return to Step 2. Otherwise, the GA is terminated and proper fuzzy logic rules are obtained by encoding the best string into a set of fuzzy rules.

4.3. The supervised learning phase

After the initial membership functions and fuzzy logic rules have been found in the first two learning phases, the whole network structure is established, and the network then enters the third learning phase to adjust the parameters of the membership functions optimally. In the parameter learning, 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 a 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, $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{y}}$, rather than the complement coded ones $\boldsymbol{x}'$ and $\boldsymbol{y}'$. 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. The learning rules can be easily extended to the multioutput case.

Layer 5: Using Eqs. (10), (16) and (17), the update 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 Eqs. (10), (16) and (17), the update 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 Eq. (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

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

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

Hence the error signal is

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

In the multioutput case, the computations in layers 5 and 4 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 in this layer. According to Eq. (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 $z_{\max} = \max(\textit{inputs of output terms node } j)$. 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 Eqs. (5), (16) and (17), the update 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

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

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

So the update 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 Eqs. (5), (16) and (17), the update 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

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

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

So the update rule of u_{ij} is

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

Finally, using Eqs. (5), (16) and (17), the update rule of sensitivity parameter γ_{ij} is derived as

$$-\frac{\partial E}{\partial \gamma_{ij}} = -\frac{\partial E}{\partial a^{(3)}}\frac{\partial a^{(3)}}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial \gamma_{ij}}, \tag{40}$$

where

$$\frac{\partial a^{(3)}}{\partial a^{(2)}} = \prod_{k\neq i} z_k^{(3)}, \tag{41}$$

$$\frac{\partial a^{(2)}}{\partial \gamma_{ij}} = \begin{cases} v_{ij} - u_{ij} & \text{if } 0 \leqslant (x_i - v_{ij})\gamma_{ij} \leqslant 1 \text{ and} \\ & \quad \text{if } 0 \leqslant (u_{ij} - x_i)\gamma_{ij} \leqslant 1, \\ v_{ij} - x_i & \text{else if } 0 \leqslant (x_i - v_{ij})\gamma_{ij} \leqslant 1, \\ x_i - u_{ij} & \text{else if } 0 \leqslant (u_{ij} - x_i)\gamma_{ij} \leqslant 1, \\ 0 & \text{otherwise.} \end{cases} \tag{42}$$

Hence, the update rule of sensitivity parameter γ_{ij} becomes

$$\gamma_{ij}(t+1) = \gamma_{ij}(t) + \eta_\gamma\frac{\partial a^{(2)}}{\partial \gamma_{ij}}\delta_i^{(3)}\prod_{k\neq i} z_k^{(3)}, \tag{43}$$

where η_γ is the learning rate of sensitivity parameter γ_{ij}.

It is noted that the performance of the FALCON trained by the learning algorithm proposed in this section highly depends on the given training data. More specifically, the completeness of learned fuzzy rules relies on the completeness of training data. The proposed learning algorithm can generate the fuzzy

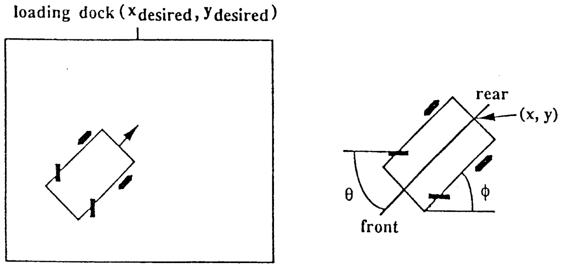


Fig. 4. Diagram of the simulated truck and loading zone.

rules that concern the input regions around training data. Hence, if a new input pattern falls for apart in these regions, the FALCON will produce a near zero-valued output, since all the learned fuzzy rules are fired weakly in this case. However, if we keep the FALCON learning, the dynamical cluster-growing ability of the fuzzy ART algorithm will create a new cluster covering this new given pattern, which in turn forms a new fuzzy rule in the FALCON. As a result, the on-line incremental learning capability of FALCON-GA keeps the rule completeness in the FALCON.

5. Illustrative examples

In this section, the proposed FALCON-GA model is applied to two simulated examples to show its performance and applicability. These two examples are the truck backing-upper problem and the chaotic time-series prediction.

Example 1 (*Control of the truck backer-upper*). Backing a truck to a loading dock is a difficult exercise. It is a non-linear control problem for which no traditional control design methods exist. Nguyen and Widrow [21] developed a neural network controller which only used numerical data for the truck backer-upper problem. Kong and Kosko [13] proposed a fuzzy controller which only used linguistic rules for the same problem. In this example, we apply the proposed FALCON-GA model as a controller to back up a simulated truck to a loading dock in a planar parking lot by learning the driving skill of experienced drivers. This FALCON-GA model enables the truck to reach the desired position successfully.

The simulated truck and loading zone are shown in Fig. 4. The truck position is exactly determined by three state variables ϕ, x and y, where ϕ is the angle of the 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 the 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 a 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 a 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-GA controller must be specified. The controller has two inputs, truck angle ϕ and the x-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 the controller is the steering angle θ. The ranges of the variables x, ϕ and θ are as follows:

$$0 \leqslant x \leqslant 100, \tag{44}$$

$$-90° \leqslant \phi \leqslant 270°, \tag{45}$$

$$-30° \leqslant \theta \leqslant 30°. \tag{46}$$

The equations of backward motion of the truck are given by

$$\begin{aligned} x(k+1) &= x(k) + d\cos\theta(k)\cos\phi(k), \\ y(k+1) &= y(k) + d\cos\theta(k)\sin\phi(k), \\ \phi(k+1) &= \tan^{-1}\left(\frac{l\sin\phi(k) + d\cos\phi(k)\sin\theta(k)}{l\cos\phi(k) - d\sin\phi(k)\sin\theta(k)}\right), \end{aligned} \tag{47}$$

where l is the length of the truck. Eq. (47) is used to obtain the next state when the present state is given.

For the purpose of training the FALCON-GA controller, learning takes place during several different tries, each starting from an initial state and terminating when the desired state is reached. In our simulation, six

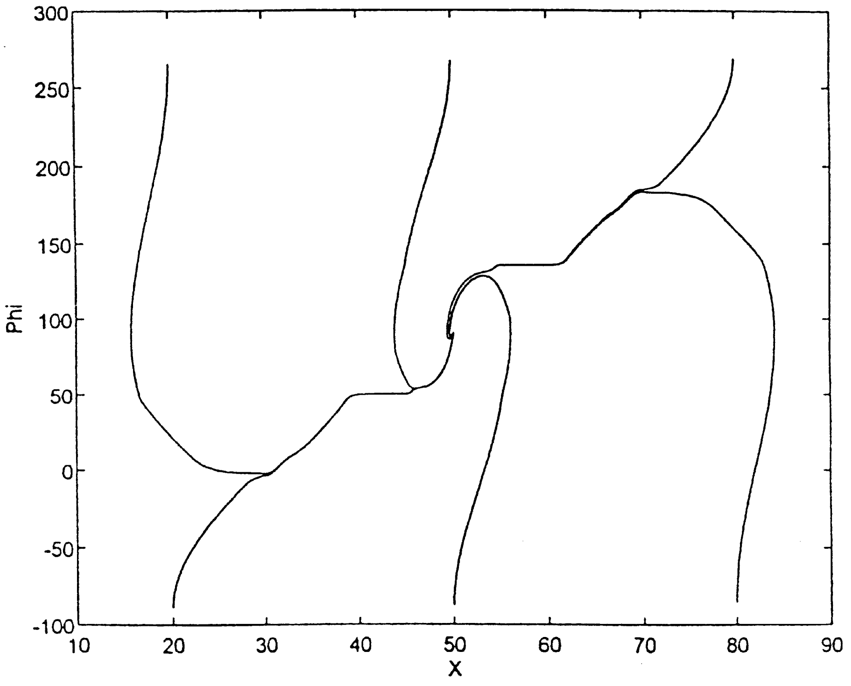


Fig. 5. The six sets of training trajectories of the truck.

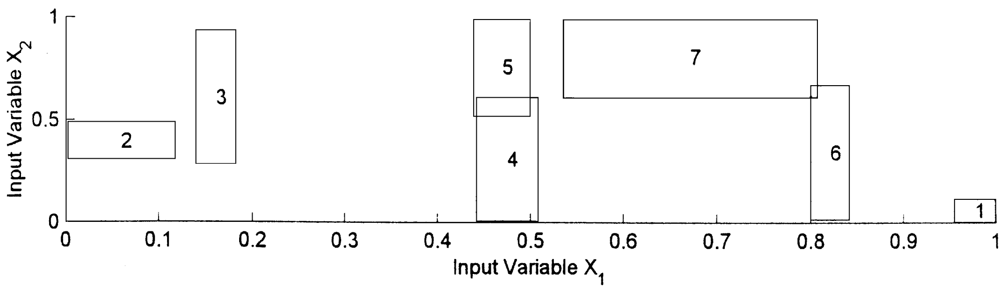


Fig. 6. Hyperboxes of the learned input membership functions in the truck backing-upper control problem.

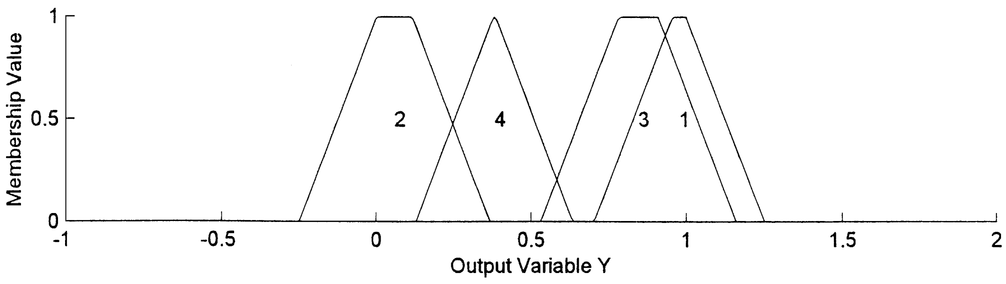


Fig. 7. Learned output membership functions in the truck backing-upper control problem.

Output \ Input	1	2	3	4	5	6	7
1	✓						
2		✓	✓		✓		
3				✓		✓	
4							✓

Fig. 8. Learned fuzzy rules in the truck backing-upper control problem.

different initial positions of the truck are chosen. The six training paths are shown in Fig. 5. The truck moves by a small fixed distance $d = 1.6$ at every step and the length of the truck is set to be $l = 1$. In the first learning phase, the initial sensitivity parameter $\gamma_{ij} = 4$ and vigilance parameter $\rho_{\text{input}} = 0.9$, $\rho_{\text{output}} = 0.8$ are chosen for the fuzzy ART algorithm. During this learning phase, all the six sets of learning trajectories are presented for partitioning the input/output spaces properly. After the input/output hyperboxes as well as the initial membership functions are located and determined, the network then enters the second learning phase to find proper fuzzy logic rules by associating input clusters and output clusters. In this phase, the GA is used to determine the links between layers 3 and 4. The population size $N_{\text{pop}} = 100$, mutation probability $p_{\text{m}} = 0.1$, and the two-point crossover operator are used. The mean number of generations in this GA learning phase is about 100 generations. After the fuzzy logic rules have been found, the whole network structure is established. The FALCON-GA controller then enters the third learning phase to adjust the parameters of the membership functions. The learning rates $\eta = 0.01$ and $\eta_\gamma = 0.5$ are chosen. The training process is continued for 600 epochs. In each epoch, all the six sets of training trajectories are presented once to the FALCON-GA controller in a random order.

After the structure/parameter learning, there are seven fuzzy logic rules generated in our simulation. The root-mean-square error of the controller with seven fuzzy rules is about 1.1°. In Fig. 6, we show the hyperboxes for the learned membership functions of two input variables. Each input membership function is two-dimensional and has a similar shape as that shown in Fig. 2(b). For convenience, in Fig. 6 we only show the "bases" of these membership functions; i.e. the parts whose membership values are 1. These membership functions overlapped one another in fact. The overlapping degree of the membership functions depends on the learned parameters γ_{ij}, which are not shown in the figure. Fig. 7 shows the learned membership functions of the output variables. Fig. 8 shows the learned seven fuzzy rules, which represent the connections from input membership functions to output membership functions. We compare the performance of the FALCON-GA model with the original FALCON-ART model proposed in Ref. [18]. The FALCON-ART is a pure neuro-fuzzy model including only two learning phases, the fuzzy ART learning phase for structure learning and the backpropagation learning phase for parameter learning. After proper learning parameters are chosen, there are nineteen fuzzy logic rules created for the FALCON-ART controller. Fig. 9 shows the learning curves for the FALCON-GA and the FALCON-ART controllers, respectively. It is observed that the performance of the FALCON-GA controller is better than that of the FALCON-ART controller. After the training process is terminated, the learned FALCON-GA is used to control the truck starting from different initial positions. Fig. 10(a)–(c) show the trajectories of the moving truck controlled by the learned FALCON-GA controller starting from the initial positions $(x, y, \phi) = (a)(10, 20, -30°), (b)(30, 20, 250°)$, and $(c)(80, 20, -30°)$. These trajectories show that the learned FALCON-GA controller can make the truck arrive at the loading dock at a right angle. This shows the good generalization capability and robustness of the FALCON-GA model.

Example 2 (*Prediction of the chaotic time-series*). Let $p(k)$, $k = 1, 2, \ldots$, be a time series. The problem of time-series prediction can be formulated as: given $p(k - m + 1), p(k - m + 2), \ldots, p(k)$, determine $p(k + l)$, where m and l are fixed positive integer; i.e. determine a mapping from $[p(k-m+1), p(k-m+2), \ldots, p(k)] \in \mathbb{R}^m$ to $[p(k + l)] \in \mathbb{R}$. The FALCON-GA 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:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1x(t), \tag{48}$$

where $\tau > 17$. In our simulation, we choose the series with $\tau = 30$. Fig. 11 shows 1000 points of this

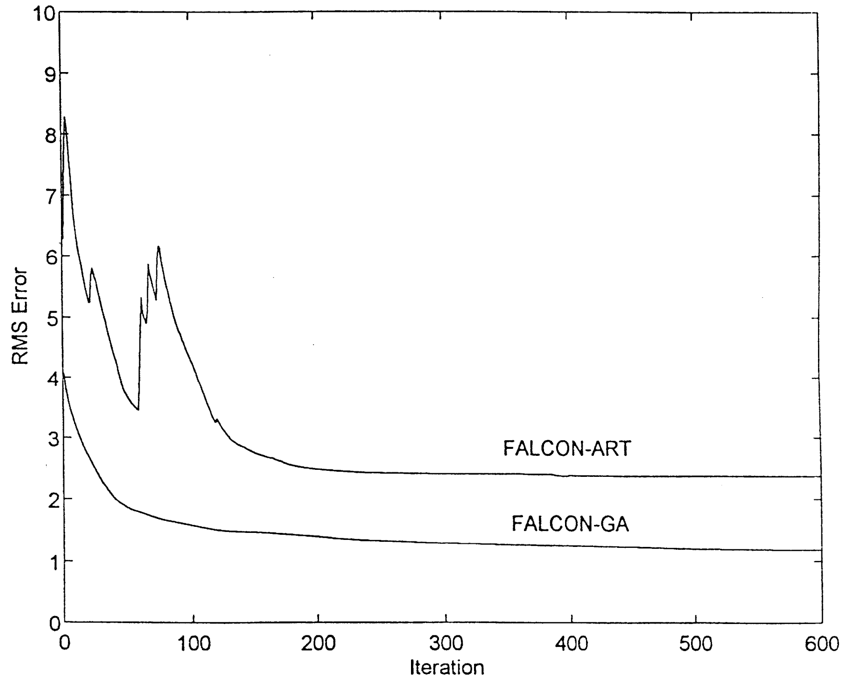


Fig. 9. Learning curves of the FALCON-GA controller and the FALCON-ART controller.

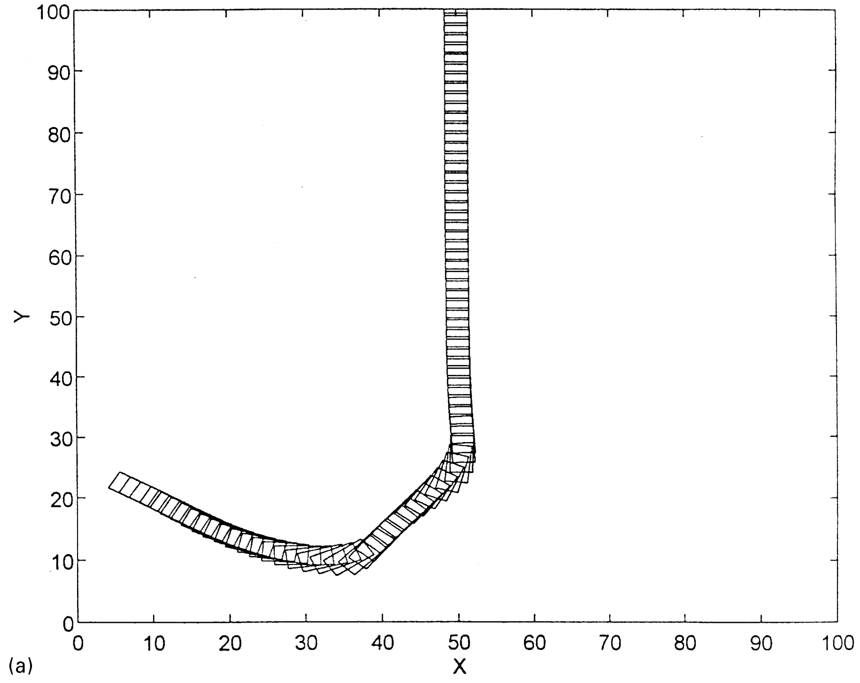
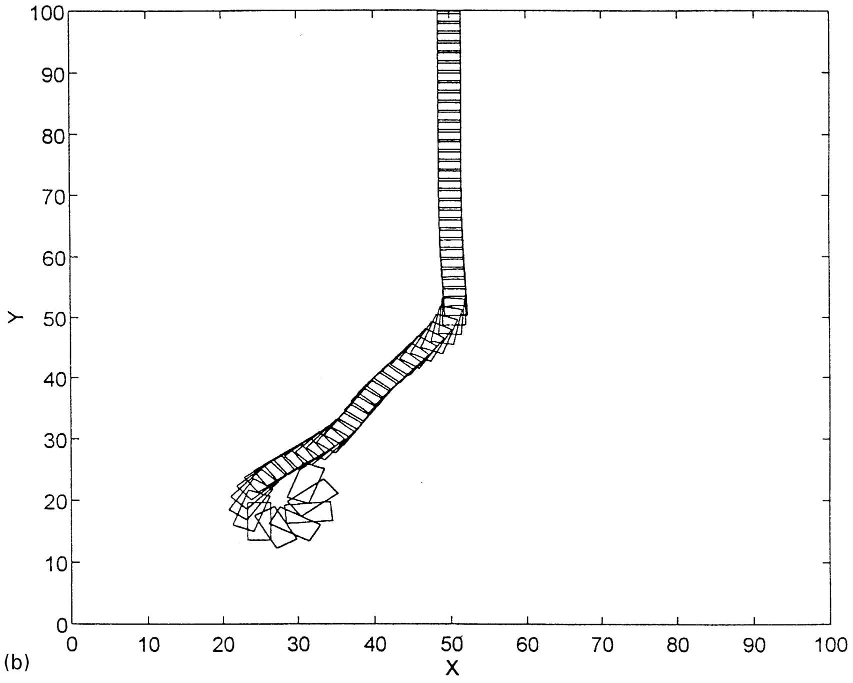
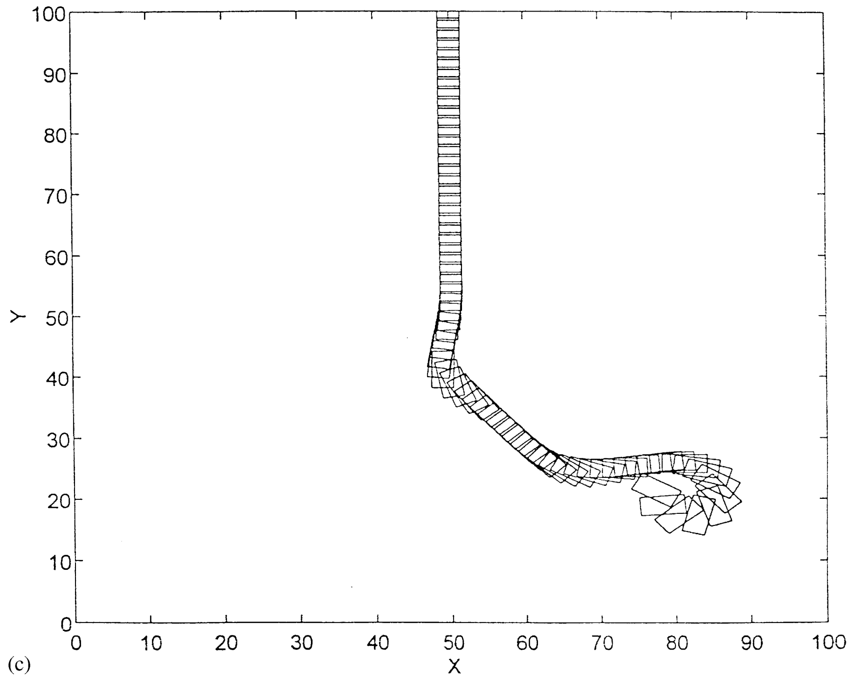


Fig. 10. Truck moving trajectories starting at three different initial positions under the control of the FALCON-GA model after learning six sets of training trajectories.



(b)



(c)

Fig. 10. Continued

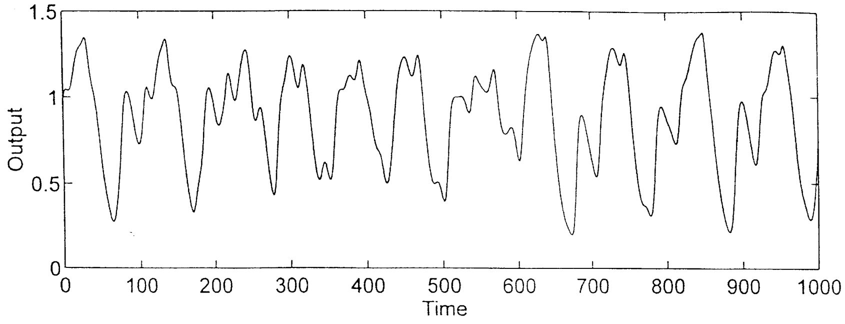


Fig. 11. The Mackey–Glass chaotic time series.

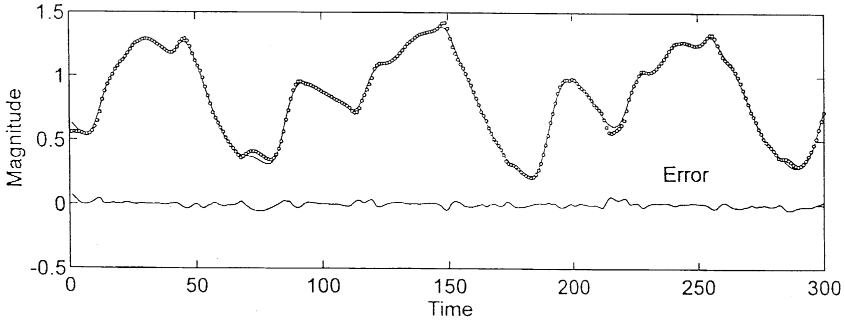


Fig. 12. The desired values (denoted by the solid line) and the predicted values (denoted by the dotted line) in Example 2. The difference between the desired and actual values is also shown in the figure, which is denoted as the solid line below the two magnitude curves.

chaotic series used to test the FALCON-GA 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 700 points of the series from $x(1)$ to $x(700)$ are used as training data, and the final 300 points from $x(701)$ to $x(1000)$ are used as test data. In the first learning phase, the initial sensitivity parameter $\gamma_{ij} = 1$ and vigilance parameter $\rho_{\text{input}} = 0.7$, $\rho_{\text{output}} = 0.6$ are chosen for the fuzzy ART algorithm. The 700 points of the series from $x(1)$ to $x(700)$ are presented for partitioning the input/output spaces properly. After the input/output hyperboxes as well as the initial membership functions are located and determined, the network then enters the second learning phase to find proper fuzzy logic rules by associating input and output clusters. In this phase, the GA is used to determine the links between layers 3 and 4. The population size $N_{\text{pop}} = 100$, mutation probability $p_{\text{m}} = 0.3$, and the two-point crossover operator are used. The mean number of generations in this GA learning phase is about 20 generations. After the fuzzy logic rules have been found, the whole network structure is established. The FALCON-GA controller then enters the third learning phase to adjust the parameters of the membership functions. The learning rates $\eta = 0.005$ and $\eta_\gamma = 0.5$ are chosen. The training process is continued for 5000 epochs. Now the training process is terminated. There are 11 fuzzy logic rules generated in our FALCON-GA model. Fig. 12 shows the prediction of the chaotic time series from $x(701)$ to $x(1000)$ when 700 training data (from $x(1)$ to $x(700)$) are used. In this figure, the predicted values of the FALCON-GA model are represented as a dotted curve and the actual values as a solid curve. The difference between the actual and predicted values is also shown in Fig. 12, which is presented as a solid curve below the two magnitude curves.

To test the generalization ability and to understand the performance of the FALCON-GA, the FALCON-ART and other approaches discussed in [18] are com-

Table 1
Performance comparison of various rule generation methods on the time series prediction problem

	FALCON-GA		FALCON-ART		Wang and Mendel	Data distribution	Kosko (AVQ) without backpropagation		Kosko (AVQ) with backpropagation			
							UCL	DCL	UCL		DCL	
Rule number (200 training data)	9	11[a]	22	30[a]	121	118	100	100	22	100	22	100
RMS error	0.03	0.0187	0.08	0.04	0.08	0.08	0.17	0.2	0.16	0.09	0.17	0.09

[a] 700 training data are used.

pared. These approaches include Wang and Mendel's approach [30] based on direct matching; the data distribution method which generates fuzzy rules according to the training data distribution in the input–output product space; the generation of Fuzzy Associative Memory (FAM) rules based on Adaptive Vector Quantization (AVQ) algorithms which contain Unsupervised Competitive Learning (UCL) and Differential Competitive Learning (DCL) proposed by Kosko [14]; and the combination of the UCL(DCL) – AVQ and backpropagation algorithm. It is noted that in [18] the training data are the 200 points of the series from $x(501)$ to $x(700)$, and the test data are the final 300 points from $x(701)$ to $x(1000)$. The generated rule numbers as well as root-mean-squares (RMS) errors of these approaches are listed in Table 1. As to the detailed construction schemes and actual predicted outputs of these models, the reader is referred to [18]. From Table 1, we find that the FALCON-GA model not only needs much fewer rules and membership functions but also achieves much smaller RMS error.

6. Conclusion

In this paper, we introduced a general connectionist model of a fuzzy logic control system, called FALCON. A structure/parameter learning algorithm, called FALCON-GA, was proposed for constructing the FALCON automatically. The proposed learning algorithm is able to partition the input and output spaces, and then find proper fuzzy rules and membership functions dynamically. Simulations showed the good generalization capability of the proposed FALCON-GA model. We also demonstrated the advantage of the integrated GA and neuro-fuzzy system over the pure neuro-fuzzy system on the local minima problem.

References

[1] D. Adler, Genetic algorithms and simulated annealing: a marriage proposal, Proc. IEEE Int. Conf. on Neural Networks, vol. II, San Franciso, CA, 1993, pp. 1104–1109.

[2] H.R. Berenji, P. Khedkar, Learning and tuning fuzzy logic controllers through reinforcements, IEEE Trans. Neural Networks 3(5) (1992) 724–740.

[3] G.A. Carpenter, S. Grossberg, D.B. Rosen, Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system, Neural Networks 4 (1991) 759–771.

[4] L. Davis, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 1991.

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

[6] J.H. Holland, Outline for a Logical Theory of Adaptive Systems, J. Assoc. Comput. Mach. 3 (1962) 297–314.

[7] J.H. Holland, Adaptation in Natural and Artificial System, University of Michigan, Ann Arbor, MI, 1975.

[8] A. Homaifar, E. McCormick, Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms, IEEE Trans. Fuzzy Systems 3(2) (1995) 129–139.

[9] H. Ishibuchi, K. Nozaki, N. Yamamoto, H. Tanaka, Selecting fuzzy if–then rules for classification problems using genetic algorithms, IEEE Trans. Fuzzy Systems 3(3) (1995) 260–270.

[10] J.S. Jang, Self-learning fuzzy controllers based on temporal back propagation, IEEE Trans. Neural Networks 3(5) (1992) 723–741.

[11] C. Karr, Genetic algorithms for fuzzy controllers, AI Expert (February, 1991) 26–33.

[12] C. Karr, Applying genetics to fuzzy logic, AI Expert (March, 1991) 39–43.

[13] S.G. Kong, B. Kosko, Comparison of fuzzy and neural truck backer upper control systems, Int. Joint Conf. on Neural Network, vol. 3, 1990, pp. 349–358.

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

[15] E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.

[16] M.A. Lee, H. Takagi, Integrating design stages of fuzzy systems using genetic algorithms, Proc. IEEE Int. Conf. on Fuzzy Systems, 1993, pp. 612–617.

[17] C.T. Lin, C.S.G. Lee, Neural-network-based fuzzy logic control and decision system, IEEE Trans. Comput. C-40(12) (1991) 1320–1336.

[18] C.J. Lin, C.T. Lin, An ART-based fuzzy adaptive learning control network, IEEE Trans. Fuzzy Systems, to appear.

[19] D.G. Luenberger, Linear and Nonlinear Programming, Addison-Wesley, Reading, MA, 1976.

[20] Z. Michalewicz, J.B. Krawezyk, A modified genetic algorithm for optimal control problems, Comput. Math. Appl. 23 (1992) 83–94.

[21] D. Nguyen, B. Widrow, The truck backer-upper: an example of self-learning in neural network, IEEE Control System Mag. 10(3) (1990) 18–23.

[22] H. Nomura, I. Hayashi, N. Wakami, A self-tuning method of fuzzy reasoning by genetic algorithm, Proc. IEEE Int. Conf. on Fuzzy Systems, 1992, pp. 236–245.

[23] V. Petridis, S. Kazarlis, A. Papaikonomou, A. Filelis, A hybrid genetic algorithm for training neural networks, in: I. Aleksander, J. Taylor (Eds.), Artificial Neural Networks 2, North-Holland, Amsterdam, 1992, pp. 953–956.

[24] P.K. Simpson, Fuzzy min–max neural networks – Part 2: Clustering, IEEE Trans. Fuzzy Systems 1(1) (1993) 32–45.

[25] M. Sugeno, T. Yasukawa, A fuzzy-logic-based approach to qualitative modeling, IEEE Trans. Fuzzy Systems 1(1) (1993) 7–31.

[26] T. Tagaki, I. Hayashi, NN-driven fuzzy reasoning, Int. J. Approx. Reasoning 5 (1991) 191–212.

[27] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Trans. System, Man, Cybernet. 15(1) (1985) 116–132.

[28] P. Thrift, Fuzzy logic synthesis with genetic algorithms, Proc. Int. Conf. Genetic Algorithms, San Diego, July 1991, pp. 509–513.

[29] L. Tsinas, B. Dachwald, A combined neural and genetic learning algorithm, Proc. IEEE Int. Conf. on Neural Networks, vol. I, 1994, pp. 770–774.

[30] L.X. Wang, J.M. Mendel, Generating fuzzy rules by learning from examples, IEEE Trans. on System, Man, Cybernet. 22(6) (1992) 1414–1427.

[31] P.J. Werbos, Neural control and fuzzy logic: connections and designs, Int. J. Approx. Reasoning 6 (1992) 185–219.

[32] R.R. Yager, Implementing fuzzy logic controller using a neural network, Fuzzy Sets and Systems 48 (1992) 53–64.