ELSEVIER

Fuzzy Sets and Systems 70 (1995) 183–212

FUZZY sets and systems

A neural fuzzy control system with structure and parameter learning

Chin-Teng Lin

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

Received June 1994

Abstract

A general connectionist model, called neural fuzzy control network (NFCN), is proposed for the realization of a fuzzy logic control system. The proposed NFCN is a feedforward multilayered network which integrates the basic elements and functions of a traditional fuzzy logic controller into a connectionist structure which has distributed learning abilities. The NFCN can be constructed from supervised training examples by machine learning techniques, and the connectionist structure can be trained to develop fuzzy logic rules and find membership functions. Associated with the NFCN is a two-phase hybrid learning algorithm which utilizes unsupervised learning schemes for structure learning and the backpropagation learning scheme for parameter learning. By combining both unsupervised and supervised learning schemes, the learning speed converges much faster than the original backpropagation algorithm. The two-phase hybrid learning algorithm requires exact supervised training data for learning. In some real-time applications, exact training data may be expensive or even impossible to obtain. To solve this problem, a reinforcement neural fuzzy control network (RNFCN) is further proposed. The RNFCN is constructed by integrating two NFCNs, one functioning as a fuzzy predictor and the other as a fuzzy controller. By combining a proposed on-line supervised structure–parameter learning technique, the temporal difference prediction method, and the stochastic exploratory algorithm, a reinforcement learning algorithm is proposed, which can construct a RNFCN automatically and dynamically through a reward-penalty signal (i.e., "good" or "bad" signal). Two examples are presented to illustrate the performance and applicability of the proposed models and learning algorithms.

Keywords: Neural networks; Connectionist; Fuzzy control; Fuzzy predictor; Gradient descent; Supervised–unsupervised learning; Reinforcement learning.

1. Introduction

Among the schemes in bringing the learning abilities of neural networks to automate and realize the design of fuzzy logic control systems [2–5, 7, 9–13, 15, 18, 19, 22, 24–27, 29], the most popular one is to imbed a fuzzy system into a neural network. In this scheme the fuzzy system is installed in an architecture isomorphic to neural networks, i.e., a multilayered network, where each node performs a function such as to make the entire network perfectly equivalent to the fuzzy system. In this approach, the gradient descent method that is akin

SSDI 0165-0114(94)00216-9

to the backpropagation algorithm is usually used to train the network. Examples include Jang's adaptive-network-based fuzzy inference system (ANFIS) [7, 22], Berenji and Khedkar's generalized approximate reasoning-based intelligent control (GARIC) [2, 3] for reinforcement learning problems, Yager's implementation of fuzzy controllers using a neural network framework [27], Nauck and Kruse's [18] fuzzy backpropagation approach [18], Wang and Mendel's [26] orthogonal least-squares learning, and many others [5, 13, 25]. The approaches developed in the paper follow this scheme.

In this paper, a general neural-network (connectionist) model is proposed to realize fuzzy logic control systems. This connectionist model, in the form of feedforward multilayer net, combines the idea of fuzzy logic controllers and neural-network structure and learning abilities into an integrated neural fuzzy control network (NFCN). An NFCN can be constructed automatically through learning from the input–output training data sets. In this connectionist structure, the input and output nodes represent the input states and output control signals, respectively, and in the hidden layers, there are nodes functioning as membership functions and fuzzy rules. The NFCN brings the spirit of human-like thinking and reasoning into a neural network structure. Moreover, NFCN performs both the parameter learning (i.e. learning the membership functions) and the structure learning (i.e., learning the fuzzy logic rules).

A structure–parameter learning algorithm is proposed for setting up the proposed NFCN. This two-phase hybrid learning algorithm combines unsupervised learning and supervised gradient-descent learning procedures to build the rule nodes and train the membership functions in two separate phases. This hybrid learning algorithm performs superiorly to the purely supervised learning algorithm (e.g., backpropagation learning rule) due to the a priori classification of training data through an overlapping receptive field before the supervised learning. Since the proposed NFCN maintains the spirit of human-like thinking and reasoning as in fuzzy logic systems, it eliminates the disadvantage of a normal feedforward multilayered net which is difficult for an outside observer to understand or to modify. So, if necessary, experts' knowledge can be easily incorporated into the proposed structure to speed up the network learning.

The two-phase hybrid learning scheme requires precise training data for setting the link weights and the link connectivity of NFCN. For some real-world applications, precise data for training/learning are usually difficult and expensive, if not impossible, to obtain. For this reason, there has been a growing interest in reinforcement learning algorithms for neural networks [1, 3, 9]. In this connection, we further extend the NFCN to the reinforcement learning problem. For the reinforcement learning problem, training data are very rough and coarse, and they are just "evaluative" as compared with the "instructive" feedback in the supervised learning problem. Training a network with this kind of evaluative feedback is called *reinforcement learning*, and this simple evaluative feedback, called *reinforcement signal*, is a scalar. In addition to the roughness and non-instructive nature of the reinforcement signal, a more challenging problem to the reinforcement learning is that a reinforcement signal may only be available at a time long after a sequence of actions has occurred. To solve the long time-delay problem, prediction capabilities are necessary in a reinforcement learning system. To achieve the goal of solving reinforcement learning problems in fuzzy logic systems, a reinforcement neural fuzzy control network (RNFCN) is proposed which consists of two closely integrated NFCNs. One NFCN, the *action network*, is used for the fuzzy controller, it can choose a proper action or decision according to the current input vector. The other NFCN, the *evaluation network*, is used as the *fuzzy predictor*, and it performs the single- or multi-step prediction of the scalar external reinforcement signal. The fuzzy predictor provides the action network with more informative and beforehand internal reinforcement signals for learning.

Associated with the proposed RNFCN is the reinforcement structure–parameter learning algorithm which uses the temporal difference technique on the evaluation network to decide the output errors for either the single- or multi-step prediction. With the knowledge of output errors, an on-line supervised structure–parameter learning algorithm is developed to train the evaluation network to obtain the proper membership functions and fuzzy logic rules. For the action network, the reinforcement structure–parameter learning algorithm allows its output nodes to perform stochastic exploration. With the

internal reinforcement signals from the fuzzy predictor, the output nodes of the action network can perform more effective stochastic searches with a higher probability of choosing a good action as well as discovering its output errors. Again, after finding the output errors, the whole action network can be trained by the proposed on-line learning algorithm. Thus, the proposed reinforcement structure–parameter learning algorithm basically utilizes the techniques of temporal difference, stochastic exploration, and the on-line supervised structure–parameter learning algorithm. It can determine the proper network size, connections, and parameters of an RNFCN dynamically through an external reinforcement signal. Moreover, learning can proceed even in the period without any external reinforcement feedback. After learning, the action network becomes an independent fuzzy logic controller which can be used to control the plant in the original environment.

In Section 2, the proposed NFCN is introduced and described. The two-phase hybrid learning algorithm is presented in Section 3. The structure of the proposed RNFCN and the corresponding reinforcement structure–parameter learning algorithm are presented in Section 4. In Section 5, two examples are used to illustrate the utility of the proposed systems. First, the model car example as suggested by Sugeno [21] is simulated to demonstrate the capabilities and the performance of the proposed hybrid learning algorithm. Then, the cart–pole balancing problem is simulated to demonstrate the capabilities of RNFCN. Finally, conclusions are summarized in Section 6.

2. Neural fuzzy control network (NFCN)

This section introduces the structure and functions of the proposed neural fuzzy control network (NFCN), which is a feedforward multilayered connectionist structure to realize the traditional fuzzy logic control systems from sets of input–output training data. The NFCN integrates the basic elements and functions of a traditional FLC (e.g., membership functions, fuzzy logic rules, fuzzification, defuzzification, and fuzzy implication) into a connectionist structure which has distributed learning abilities to learn the input/output membership functions and fuzzy logic rules. Fig. 1 shows the structure of our proposed NFCN. The system has five layers. Nodes at layer one are input nodes (*linguistic nodes*) which represent input linguistic variables. Layer five is the output layer. Nodes at layers two and four are *term nodes* and act as membership functions to represent the terms of each 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. Therefore, for each rule node, there is at most one link (perhaps none) from some term node of a linguistic node. This is true both for precondition links and consequent links. The links at layers two and five are fully connected between linguistic nodes and their corresponding term nodes. The arrow on the link indicates the normal signal flow direction when this network is in use. We shall later indicate the signal propagation, layer by layer, according to the arrow direction. Signals may flow in the reverse direction in the learning process as discussed later.

With this five-layered structure of the proposed connectionist model, the basic functions of a node can be defined. A typical network consists of a unit which has some finite fan-in of connections represented by weight values from other units and fan-out of connections to other units. Associated with the fan-in of a unit is an integration function f which serves to combine information, activation, or evidence from other nodes. This function provides the net input for this node:

$$\text{net-input} = f(u_1^{(k)}, u_2^{(k)}, \ldots, u_p^{(k)}; w_1^{(k)}, w_p^{(k)}, \ldots, w_p^{(k)}), \tag{1}$$

where $u_i^{(k)}$ represents the ith input signal at the kth layer, $w_i^{(k)}$ represents the ith link weight of the kth layer, the superscript k indicates the layer number, and p represents the number of a node's input connections. This

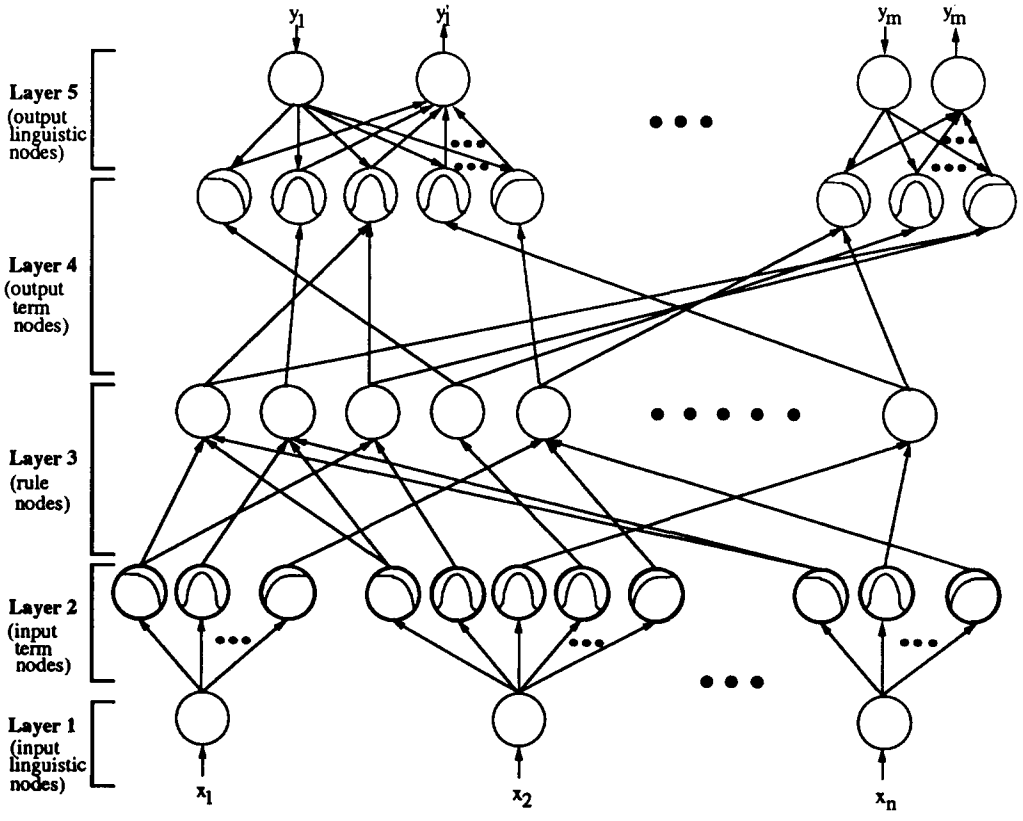

Fig. 1. Proposed neural fuzzy control network (NFCN).

notation will also be used in the following equations. A second action of each node is to output an activation value as a function of its net input:

$$\text{output} = o_i^{(k)} = a(f), \tag{2}$$

where $a(\cdot)$ denotes the activation function. For example (in standard form),

$$f = \sum_{i=1}^{p} w_i^{(k)} u_i^{(k)} \quad \text{and} \quad a = \frac{1}{1 + e^{-f}}. \tag{3}$$

We shall next describe the functions of the nodes in each of the five layers of the proposed connectionist model.

Layer 1: The nodes in this layer transmit input values directly to the next layer. That is,

$$f = u_i^{(1)} \quad \text{and} \quad a = f. \tag{4}$$

From Eq. (4), the link weight at layer one ($w_i^{(1)}$) is unity.

Layer 2: Each node in this layer functions as a membership function. For example, for a bell-shaped function, we have

$$f = M_{x_i}^{j}(m_{ij}, \sigma_{ij})^2 = -\frac{(u_i^{(2)} - m_{ij})^2}{\sigma_{ij}^2} \quad \text{and} \quad a = e^f, \tag{5}$$

where m_{ij} and σ_{ij} are, respectively, the center (or mean) and the width (or variance) of the bell-shaped function of the jth term of the ith input linguistic variable x_i. We shall use bell-shaped membership function in this paper.

Layer 3: The links in this layer are used to perform precondition matching of fuzzy logic rules. Hence, the rule nodes should perform the fuzzy AND operation,

$$f = \min(u_1^{(3)}, u_2^{(3)}, \ldots, u_p^{(3)}) \quad \text{and} \quad a = f. \tag{6}$$

The link weight in layer three ($w_i^{(3)}$) is then unity. We can also use product operator for the fuzzy AND operation. It is differentiable and suitable for the derivation of a learning algorithm. However, it requires more computations than the min operator.

Layer 4: For the hybrid learning algorithm, the nodes in this layer have two operation modes: *down–up* transmission and *up–down* transmission modes. In the down–up transmission mode, signals flow upwards and the links at layer four perform the fuzzy OR operation to integrate the fired rules which have the same consequent, where the bounded sum operation is used for fuzzy OR:

$$f = \sum_{i=1}^{p} u_i^{(4)} \quad \text{and} \quad a = \min(1, f). \tag{7}$$

Hence, the link weight $w_i^{(4)} = 1$. In the up–down transmission mode, signals flow downwards and the nodes in this layer and the links in layer five function exactly the same as those in layer two. For the RNFCN, the nodes in this layer only operate in the down–up transmission mode.

Layer 5: For the hybrid learning algorithm, there are two kinds of nodes in this layer. The first kind of node performs the up–down transmission for the training data y_i to feed into the network. For this kind of node,

$$f = y_i \quad \text{and} \quad a = f. \tag{8}$$

The second kind of node performs the down–up transmission for the decision signal y_i' output. These nodes and the layer-five links attached to them act as the defuzzifier. If m_{ij}'s and σ_{ij}'s are the centers and the widths of output membership functions, respectively, then the following functions can be used to simulate the *center of area* defuzzification method [8]:

$$f = \sum w_{ij}^{(5)} u_i^{(5)} = \sum (m_{ij}\sigma_{ij}) u_i^{(5)} \quad \text{and} \quad a = \frac{f}{\sum \sigma_{ij} u_i^{(5)}}. \tag{9}$$

Here the link weight at layer five ($w_{ij}^{(5)}$) is $m_{ij}\sigma_{ij}$. For the RNFCN, we have only one kind of nodes performing the down–up transmission for the decision signal output.

Based on the above connectionist structure, a two-phase hybrid learning scheme will be proposed to realize the NFCN from the given input–output training data sets. This learning algorithm includes structure and parameter learning to determine optimal centers (m_{ij}'s) and widths (σ_{ij}'s) of term nodes in layers two and four. Also, it will learn fuzzy logic rules by deciding the connection types of the links at layers three and four; that is, the precondition links and consequent links of the rule nodes.

3. Two-phase hybrid learning algorithm

In this section, we shall present the proposed two-phase learning scheme for realizing the NFCN from the given input–output training data sets. In phase one, a self-organized learning scheme is used to locate initial input and output membership functions and to find the presence of fuzzy logic rules. In phase two, a supervised learning scheme is used to adjust optimally both input and output membership functions for

desired outputs. Before this network is trained, an initial form of the network is first constructed. Then, during the learning process, some nodes and links of this initial network are deleted or combined to form the final structure of the network. Let $T(x_i)$ denote the term set of the linguistic variable x_i. Then in its initial form (see Fig. 1), there are $\prod_i |T(x_i)|$ rule nodes with the inputs of each rule node coming from one possible combination of the terms of input linguistic variables under the constraint that only one term in a term set can be a rule node's input. Here $|T(x_i)|$ denotes the number of terms of x_i (i.e., the number of fuzzy partitions of input state linguistic variable x_i). So the state space is initially divided into $|T(x_1)| \times |T(x_2)| \times \cdots \times |T(x_n)|$ linguistically defined nodes (or fuzzy cells) which represent the preconditions of fuzzy rules. Also, initially, the links between the rule nodes and the output term nodes are fully connected, meaning that the consequents of the rule nodes are not yet decided. Only a suitable term in each output linguistic variable's term set will be chosen after the learning process.

3.1. *Self-organized learning phase*

The problem for the self-organized learning can be stated as: Given the training input data $x_i(t), i = 1, \ldots, n$, the desired output value $y_i(t)$, $i = 1, \ldots, m$, the fuzzy partitions $|T(x)|$ and $|T(y)|$, and the desired shapes of membership functions, we want to locate the membership functions and find the fuzzy logic rules. In this phase, the network works in a two-sided manner; that is, the nodes and links at layer four are in the up–down transmission mode so that the training input and output data are fed into this network from both sides.

First, the centers (or means) and the widths (or variances) of the (input and output) membership functions are determined by self-organized learning techniques analogous to statistical clustering. This serves to allocate network resources efficiently by placing the domains of membership functions covering only those regions of the input/output space where data are present. Kohonen's feature-maps algorithm [14] is adopted here to find the center m_i of the membership function:

$$\| x(t) - m_{\text{closest}}(t) \| = \min_{1 \leq i \leq k} \{ \| x(t) - m_i(t) \| \}, \tag{10}$$

$$m_{\text{closest}}(t+1) = m_{\text{closest}}(t) + \alpha(t)[x(t) - m_{\text{closest}}(t)], \tag{11}$$

$$m_i(t+1) = m_i(t) \quad \text{for } m_i \neq m_{\text{closest}}, \tag{12}$$

where $x(t)$ is the value of some linguistic variable at time t, $m_i(t)$ is the center of cluster i at time t, $m_{\text{closest}}(t)$ is the center closest to $x(t)$, $\alpha(t)$ is a monotonically decreasing scalar learning rate, and $k = |T(x)|$. This adaptive formulation runs independently for each input and output linguistic variable. The determination of which of the m_i's is m_{closest} can be accomplished in constant time via a winner-take-all circuit. Once the centers of membership functions are found, their widths can be simply determined by the *first-nearest-neighbor* heuristic as

$$\sigma_i = \frac{|m_i - m_{\text{closest}}|}{r}, \tag{13}$$

where r is an overlap parameter. In the second learning phase, the centers and the widths of the membership functions will be fine-tuned optimally.

After the parameters of the membership functions have been found, the signals from both external sides can reach the output points of term nodes at layers two and four (see Fig. 1). Furthermore, the outputs of term nodes at layer two can be transmitted to rule nodes through the initial architecture of layer-three links. So we can get the firing strength of each rule node. Based on these rule firing strengths (denoted as $o_i^{(3)}(t)$'s) and the outputs of term nodes at layer four (denoted as $o_j^{(4)}(t)$'s), we want to decide the correct consequent links (layer-four links) of each rule node to find the existing fuzzy logic rule by competitive learning

algorithms [15]. As stated in the last section, the links at layer four are initially fully connected. We denote the weight of the link between the ith rule node and the jth output term node as w_{ij}. The following competitive learning law is used to update these weights for each training data set:

$$\dot{w}_{ij}(t) = o_j^{(4)}(-w_{ij} + o_i^{(3)}). \tag{14}$$

Here $o_j^{(4)}$ serves as a win–loss index of the jth term node at layer four. The theme of this law is that *learn if win*. In the extreme case, if $o_j^{(4)}$ is a 0–1 threshold function, then the above law says *learn only if win*.

After the competitive learning through the whole training data set, the link weights at layer four represent the strength of the existence of the corresponding rule consequent. Among the links which connect a rule node and the term nodes of an output linguistic node, at most one link with maximum weight is chosen and the others are deleted. Hence, only one term in an output linguistic variable's term set can become one of the consequents of a fuzzy logic rule. If all the link weights between a rule node and the term nodes of an output linguistic node are very small, then all the corresponding links are deleted, meaning that this rule node has little or no relation to this output linguistic variable. If all the links between a rule node and the layer-four nodes are deleted, then this rule node can be eliminated since it does not affect the outputs.

After the consequents of rule nodes are determined, the rule combination is performed to reduce the number of rules. The criteria for a set of rule nodes to be combined into a single rule node are (1) they have exactly the same consequents, (2) some preconditions are common to all the rule nodes in this set, and (3) the union of other preconditions of these rule nodes composes the whole term set of some input linguistic variables. If a set of nodes meets these criteria, a new rule node with only the common preconditions can replace this set of rule nodes. An example is illustrated in Fig. 2, where the first term of x_0 is common to all the rules and it thus becomes the only precondition of the resultant rule after the rule combination scheme.

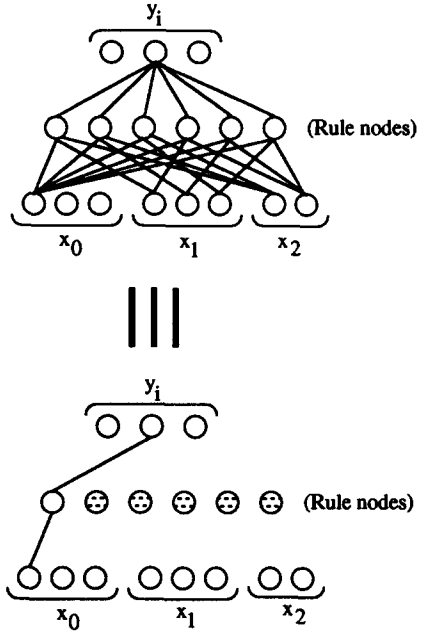

Fig. 2. Example of combination of rule nodes.

3.2. Supervised learning phase

After the fuzzy logic rules have been found, the whole network structure is established, and the network then enters the second learning phase to adjust the parameters of the membership functions optimally. The problem for the supervised learning can be stated as: Given the training input data $x_i(t)$, $i = 1, \ldots, n$, the desired output value $y_i(t)$, $i = 1, \ldots, m$, the fuzzy partitions $|T(x)|$ and $|T(y)|$, and the fuzzy logic rules, adjust the parameters of the membership functions optimally. These fuzzy logic rules were learned in the first-phase learning or, in some application domains, they can be given by experts. In the second-phase learning, the network works in the feedforward manner; that is, the nodes and the links at layer four are in the down–up transmission mode. The idea of backpropagation is used for this supervised 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 each training data set, 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. Assuming that w is the adjustable parameter in a node (e.g., the center of a membership function), the general learning rule used is

$$\Delta w \propto -\partial E/\partial w, \tag{16}$$

$$w(t+1) = w(t) + \eta\left(-\frac{\partial E}{\partial w}\right), \tag{17}$$

where η is the learning rate, and

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial(\text{activation})}\frac{\partial(\text{activation})}{\partial w} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial w}. \tag{18}$$

To show the learning rule, we shall show the computations of $\partial E/\partial w$, layer by layer, starting at the output nodes, and we will use the bell-shaped membership functions with centers m_i's and widths σ_i's as the adjustable parameters for these computations.

Layer 5: Using Eqs. (18) and (9), the adaptive rule of the center m_i is derived as

$$\frac{\partial E}{\partial m_i} = \frac{\partial E}{\partial a^{(5)}}\frac{\partial a^{(5)}}{\partial m_i} = -[y(t) - \hat{y}(t)]\frac{\sigma_i u_i^{(5)}}{\sum \sigma_i u_i^{(5)}}. \tag{19}$$

Hence, the center parameter is updated by

$$m_i(t+1) = m_i(t) + \eta[y(t) - \hat{y}(t)]\frac{\sigma_i u_i^{(5)}}{\sum \sigma_i u_i^{(5)}}. \tag{20}$$

Similarly, using Eqs. (18) and (9), the adaptive rule of the width σ_i is derived as

$$\frac{\partial E}{\partial \sigma_i} = \frac{\partial E}{\partial a^{(5)}}\frac{\partial a^{(5)}}{\partial \sigma_i} = -[y(t) - \hat{y}(t)]\frac{m_i u_i^{(5)}(\sum \sigma_i u_i^{(5)}) - (\sum m_i \sigma_i u_i^{(5)})u_i^{(5)}}{(\sum \sigma_i u_i^{(5)})^2}. \tag{21}$$

Hence, the width parameter is updated by

$$\sigma_i(t+1) = \sigma_i(t) + \eta[y(t) - \hat{y}(t)]\frac{m_i u_i^{(5)}(\sum \sigma_i u_i^{(5)}) - (\sum m_i \sigma_i u_i^{(5)})u_i^{(5)}}{(\sum \sigma_i u_i^{(5)})^2}. \tag{22}$$

The error to be propagated to the preceding layer is

$$\delta^{(5)} = \frac{-\partial E}{\partial a^{(5)}} = y(t) - \hat{y}(t). \tag{23}$$

Layer 4: In the down–up mode, there is no parameter to be adjusted in this layer. Only the error signals ($\delta_i^{(4)}$'s) need to be computed and propagated. The error signal $\delta_i^{(4)}$ is derived as in the following:

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

where (from Eq. (9))

$$\frac{\partial a_i^{(5)}}{\partial a_i^{(4)}} = \frac{\partial a^{(5)}}{\partial u_i^{(5)}} = \frac{m_i\sigma_i(\sum \sigma_i u_i^{(5)}) - (\sum m_i\sigma_i u_i^{(5)})\sigma_i}{(\sum \sigma_i u_i^{(5)})^2} \tag{25}$$

and from Eq. (23),

$$\frac{\partial E}{\partial a^{(5)}} = -\delta^{(5)} = -[y(t) - \hat{y}(t)]. \tag{26}$$

Hence, the error signal is

$$\delta_i^{(4)}(t) = [y(t) - \hat{y}(t)] \frac{m_i\sigma_i(\sum \sigma_i u_i^{(5)}) - (\sum m_i\sigma_i u_i^{(5)})\sigma_i}{(\sum \sigma_i u_i^{(5)})^2}. \tag{27}$$

In the multiple-output case, the computations in layers five and four are exactly the same as the above and proceed independently for each output linguistic variable.

Layer 3: As in layer four, only the error signals need to be computed. According to Eq. (7), this error signal can be derived as

$$-\delta_i^{(3)} = \frac{\partial E}{\partial a_i^{(3)}} = \frac{\partial E}{\partial a_i^{(4)}} \frac{\partial a_i^{(4)}}{\partial a_i^{(3)}} = -\delta_i^{(4)} \frac{\partial a_i^{(4)}}{\partial u_i^{(4)}} = -\delta_i^{(4)}. \tag{28}$$

Hence, the error signal is $\delta_i^{(3)} = \delta_i^{(4)}$. If there are multiple outputs, then the error signal becomes $\delta_i^{(3)} = \sum_k \delta_k^{(4)}$, 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. (18) and (5), the adaptive rule of m_{ij} is derived as in the following:

$$\frac{\partial E}{\partial m_{ij}} = \frac{\partial E}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial m_{ij}} = \frac{\partial E}{\partial a_i^{(2)}} e^{f_i} \frac{2(u_i^{(2)} - m_{ij})}{\sigma_{ij}^2}, \tag{29}$$

where (from Eq. (28))

$$\frac{\partial E}{\partial a_i^{(2)}} = \sum_k \frac{\partial E}{\partial a_i^{(3)}} \frac{\partial a_i^{(3)}}{\partial a_i^{(2)}}, \tag{30}$$

$$\frac{\partial E}{\partial a_i^{(3)}} = -\delta_k^{(3)}, \tag{31}$$

and from Eq. (6),

$$\frac{\partial a_i^{(3)}}{\partial a_i^{(2)}} = \frac{\partial a_i^{(3)}}{\partial u_i^{(3)}} = \begin{cases} 1 & \text{if } u_i^{(3)} = \min(\text{inputs of rule node } k), \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

Hence,

$$\frac{\partial E}{\partial a_i^{(2)}} = \sum_k q_k, \tag{33}$$

where the summation is performed over the rule nodes that $a_i^{(2)}$ feeds into, and

$$q_k = \begin{cases} -\delta_k^{(3)} & \text{if } a_i^{(2)} \text{ is minimum in the } k\text{th rule node's inputs,} \\ 0 & \text{otherwise.} \end{cases} \tag{34}$$

So the adaptive rule of m_{ij} is

$$m_{ij}(t+1) = m_{ij}(t) - \eta \frac{\partial E}{\partial a_i^{(2)}} e^{f_i} \frac{2(u_i^{(2)} - m_{ij})}{\sigma_{ij}^2}. \tag{35}$$

Similarly, using Eqs. (18), (5), and (30)–(34), the adaptive rule of σ_{ij} is derived as

$$\frac{\partial E}{\partial \sigma_{ij}} = \frac{\partial E}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial \sigma_{ij}} = \frac{\partial E}{\partial a_i^{(2)}} e^{f_i} \frac{2(u_i^{(2)} - m_{ij})^2}{\sigma_{ij}^3}. \tag{36}$$

Hence, the adaptive rule of σ_{ij} becomes

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) - \eta \frac{\partial E}{\partial a_i^{(2)}} e^{f_i} \frac{2(u_i^{(2)} - m_{ij})^2}{\sigma_{ij}^3}. \tag{37}$$

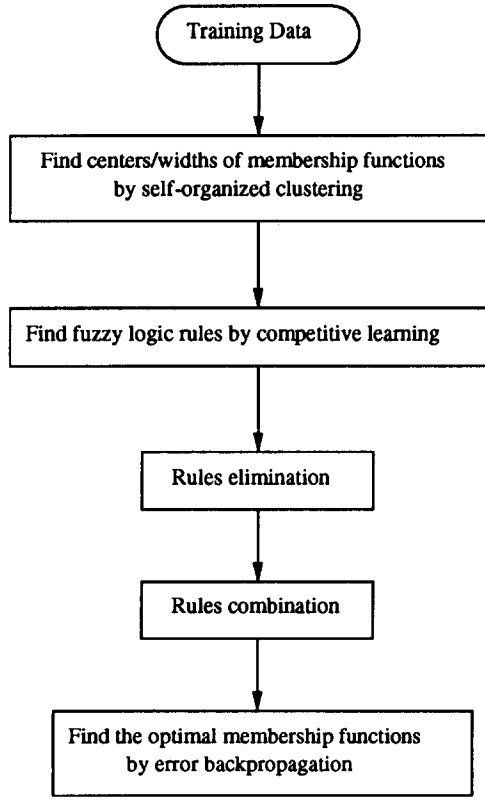

Fig. 3. Flow chart of the proposed hybrid learning algorithm.

The proposed two-phase hybrid learning procedure is summarized by the flow chart in Fig. 3. The convergence speed of the backpropagation in the phase-two learning is found superior to the normal backpropagation scheme since the self-organized learning process in phase one has done much of the learning work in advance. The hybrid learning algorithm requires that sets of precise input and output training data are available. In the next section, we shall consider a more difficult learning problem, the reinforcement learning problem.

4. Reinforcement structure–parameter learning algorithm for RNFCN

Unlike the supervised learning problem in which the correct "target" output values are given for each input pattern to instruct the network's learning, the reinforcement learning problem has only very simple "evaluative" or "critic" information instead of "instructive" information available for learning. In the extreme case, there is only a single bit of information to indicate whether the output is right or wrong. Fig. 4 shows how a network and its training environment interact in a reinforcement learning problem. The environment supplies a time-varying vector of input to the network, receives its time-varying vector of output/actions, and then provides a time-varying scalar reinforcement signal. In this paper, the reinforcement signal $r(t)$ can be one of the following forms: (1) a two-valued number, $r(t) \in \{-1, 1\}$, such that $r(t) = 1$ means "a success" and $r(t) = -1$ means "a failure"; (2) a multi-valued discrete number in the range $[-1, 1]$, for example, $r(t) \in \{-1, -0.5, 0, 0.5, 1\}$ which corresponds to different discrete degrees of failure or success; or (3) a real number, $r(t) \in [-1, 1]$, which represents a more detailed and continuous degree of failure or success. We also

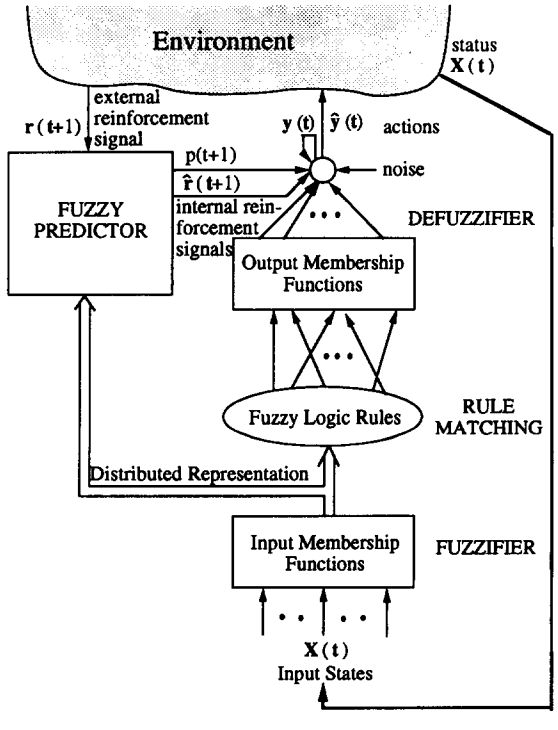

Fig. 4. Proposed reinforcement neural fuzzy control network (RNFCN).

assume that $r(t)$ is the reinforcement signal available at time step t and is caused by the input and actions chosen at time step $t-1$ or even affected by earlier input and actions. The objective of learning is to maximize a function of this reinforcement signal, such as the expectation of its value on the upcoming time step or the expectation of some integral of its values over all future time.

To resolve the reinforcement learning problems, a new structure, called the reinforcement neural fuzzy control network (RNFCN), is proposed. The proposed RNFCN, as shown in Fig. 4, integrates two NFCNs into a learning system: one NFCN for the fuzzy controller and the other NFCN for the fuzzy predictor. These two NFCNs share the same layers 1 and 2 and have individual layer 3 to layer 5, which are not clearly shown in the fuzzy predictor in Fig. 4. Each network has exactly the same structure as shown in Fig. 1. In other words, the fuzzy controller (action network) and the fuzzy predictor (evaluation network) share the same distributed representation of input states by using the same input membership functions (i.e., the same fuzzifier), but they have independent fuzzy logic rules (a different rule base and decision-making process) and different output membership functions (a different defuzzifier). The action network can have multiple outputs as shown in Fig. 1, although only one output node is shown in Fig. 4. In the multi-output case, all the output nodes of the action network receive the same internal reinforcement signals from the evaluation network. The evaluation network has only one output node since it is used to predict the external scalar reinforcement signal. The action network decides a best action to impose onto the environment in the next time step according to the current environment status. The evaluation network models the environment such that it can perform a single- or multi-step prediction of the reinforcement signal that will eventually be obtained from the environment for the current action chosen by the action network. The predicted reinforcement signal can provide the action network beforehand as well as more detailed reward/penalty information ("internal reinforcement signals") about the candidate action for the action network to learn and to decrease the uncertainty it faces to speed up the learning. We shall now describe the details of this reinforcement learning algorithm in the following subsections.

4.1. Stochastic exploration

In this subsection, we first develop the learning algorithm for the action network. The goal of the reinforcement structure–parameter learning algorithm is to adjust the parameters (e.g., m_i's) of the action network, to change the connectionist structure or even to add new nodes, if necessary, such that the reinforcement signal is maximum; that is,

$$\Delta m_i \propto \partial r/\partial m_i. \tag{38}$$

To determine $\partial r/\partial m_i$, we need to know $\partial r/\partial y$, where y is the output of the action network. (For clarity, we discuss the single-output case first.) Since the reinforcement signal does not provide any hint as to what the right answer should be in terms of a cost function, there is no gradient information. Hence, the gradient $\partial r/\partial y$ can only be estimated. If we can estimate $\partial r/\partial y$, then an on-line supervised structure–parameter learning algorithm can be directly derived for the action network to solve the reinforcement learning problem. To estimate the gradient information in a reinforcement learning network, there needs to be some source of randomness in the manner in which output actions are chosen by the action network such that the space of possible output can be explored to find a correct value. Thus, the output nodes (layer 5) of the action network are now designed to be stochastic units which compute their output as a stochastic function of their input. The functions of nodes in the other layers of the action network remain unchanged as described in Section 2. Such an approach has also been used in other reinforcement learning algorithms [1, 9] and is consistent with the closely related theory of stochastic learning automata [17].

In our learning algorithm, the gradient information $\partial r/\partial y$ is also estimated by the stochastic exploratory method. In particular, the intuitive idea behind the multi-parameter distributions is used for the stochastic search of network output units. In estimating the gradient information, the output y of the action network

does not directly act on the environment. Instead, it is treated as a mean (expected) action. The actual action $\hat{y}$ is chosen by exploring a range around this mean point. This range of exploration corresponds to the variance of a probability function which is the normal distribution in our design. This amount of exploration $\sigma(t)$ is chosen as

$$\sigma(t) = \frac{k}{2}[1 - \tanh(p(t))] = \frac{k}{1 + e^{2p(t)}}, \tag{39}$$

where k is a search-range scaling constant which can be simply set to 1, and $p(t)$ is the predicted (expected) reinforcement signal used to predict $r(t)$. Eq. (39) is a monotonic decreasing function between k and 0, and $\sigma(t)$ can be interpreted as the extent to which the output node searches for a better action. Since $p(t)$ is the expected reward signal, if $p(t)$ is small, the exploratory range $\sigma(t)$ will be large according to Eq. (39). On the contrary, if $p(t)$ is large, $\sigma(t)$ will be small. This amounts to narrowing the search about the mean $y(t)$ if the expected reinforcement signal is large. This can provide a higher probability to choose an actual action $\hat{y}(t)$ which is very close to $y(t)$, since it is expected that the mean action $y(t)$ is very close to the best action possible for the current given input vector. On the other hand, the search range about the mean $y(t)$ is broadened if the expected reinforcement signal is small such that the actual action can have a higher probability of being quite different from the mean action $y(t)$. Thus, if an expected action has a smaller expected reinforcement signal, we can have more novel trials. In terms of searching, the use of multi-parameter distributions in the stochastic nodes (the output nodes of the action network) could allow independent control of the location being searched and the breadth of the search around that location. In the above two-parameter distribution approach, a predicted reinforcement signal is necessary to decide the search range $\sigma(t)$. This predicted reinforcement signal can be obtained from the fuzzy predictor. If no such prediction is available, the search range $\sigma(t)$ can be set as a constant. Then the multi-parameter distribution approach reduces to the single-parameter distribution approach, which has been widely used in the reinforcement learning algorithms [1]. Once the variance has been decided, the actual output of the stochastic node can be set as

$$\hat{y}(t) = N(y(t), \sigma(t)). \tag{40}$$

That is, $\hat{y}(t)$ is a normal or Gaussian random variable with the density function

$$f(\hat{y}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(\hat{y}-y)^2/2\sigma^2}. \tag{41}$$

For a real-world application, $\hat{y}(t)$ should be properly scaled to the final output to fit the input specifications of the controlled plant. This scaling factor or method is application-oriented.

The gradient information is estimated as

$$\frac{\partial r}{\partial y} \approx [r(t) - p(t)]\left[\frac{\hat{y}(t-1) - y(t-1)}{\sigma(t-1)}\right] \equiv [r(t) - p(t)]\left[\frac{\hat{y} - y}{\sigma}\right]_{t-1}, \tag{42}$$

where the subscript $t-1$ represents the time displacement and σ is a scaling factor. The time displacements in Eq. (42) and the following equations reflect the assumption that the reinforcement signal (which may be the "predicted" reinforcement signal in the multi-step fuzzy predictor) at time step t depends on the input and actions chosen at time step $t-1$. In Eq. (42), the term $(\hat{y} - y)/\sigma$ is the normalized difference between the actual and the expected actions, $r(t)$ is the real reinforcement feedback for the actual action $\hat{y}(t-1)$, and $p(t)$ is the predicted reinforcement signal for the expected action $y(t-1)$. Eq. (42) was derived based on the following intuitive concept. If $r(t) > p(t)$, then $\hat{y}(t-1)$ is a better action than the expected one, $y(t-1)$, and $y(t-1)$ should be moved closer to $\hat{y}(t-1)$. If $r(t) < p(t)$, then $\hat{y}(t-1)$ is a worse action than the expected one, and $y(t-1)$ should be moved farther away from $\hat{y}(t-1)$. This idea also comes from the observations of a discrete gradient descent method. The concept behind Eq. (42) is frequently adopted in the stochastic exploration techniques.

After the gradient information is available, we have transformed the reinforcement learning problem to the supervised learning problem and can apply the gradient descent method to develop the reinforcement structure–parameter learning algorithm for the action network in the proposed RNFCN. According to Fig. 5, after the initialization process the learning algorithm enters the training loop in which each loop corresponds to an incoming internal reinforcement signal. The goal now is to maximize the reinforcement signal $r(t)$. For each input vector from the environment, starting at the input nodes, a forward pass computes the activity levels of all the nodes in the network, and at the end, stochastic exploration is performed at the output node to predict $\partial r/\partial y$. Then, starting at the output nodes, a backward pass computes $\partial r/\partial f$ for all the hidden nodes. Assuming that w is an adjustable parameter in a node (e.g., the center of a membership function), the general parameter learning rule used is

$$\Delta w \propto \partial r/\partial w, \tag{43}$$

$$w(t+1) = w(t) + \eta(\partial r/\partial w), \tag{44}$$

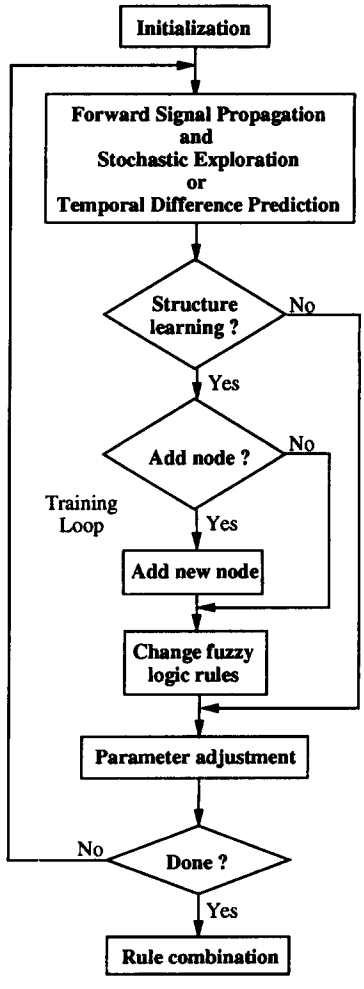

Fig. 5. Flow chart of the proposed reinforcement structure/parameter learning algorithm.

where η is the learning rate, and

$$\frac{\partial r}{\partial w} = \frac{\partial r}{\partial(\text{activation})} \frac{\partial(\text{activation})}{\partial w} = \frac{\partial r}{\partial a} \frac{\partial a}{\partial w}. \tag{45}$$

To show the learning rule, we shall show the computations of $\partial r/\partial w$, layer by layer, starting from the output layer; we use the bell-shaped membership functions with centers m_i's and widths σ_i's as the adjustable parameters for these computations.

Layer 5: Using Eqs. (45), (42) and (9), the adaptive rule of the center m_i is derived:

$$\frac{\partial r}{\partial m_i} = \frac{\partial r}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial m_i} = [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1} \left[\frac{\sigma_i u_i^{(5)}}{\sum \sigma_i u_i^{(5)}} \right]_{t-1}. \tag{46}$$

Hence, the expected updated amount of the center parameter is

$$\Delta m_i(t) = \eta [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1} \left[\frac{\sigma_i u_i^{(5)}}{\sum \sigma_i u_i^{(5)}} \right]_{t-1}. \tag{47}$$

Similarly, using Eqs. (45), (42), and (9), the adaptive rule of the width σ_i is derived:

$$\frac{\partial r}{\partial \sigma_i} = \frac{\partial r}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial \sigma_i} = [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1} \left[\frac{m_i u_i^{(5)} (\sum \sigma_i u_i^{(5)}) - (\sum m_i \sigma_i u_i^{(5)}) u_i^{(5)}}{(\sum \sigma_i u_i^{(5)})^2} \right]_{t-1}. \tag{48}$$

Hence, the expected updated amount of the width parameter is

$$\Delta \sigma_i(t) = \eta [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1} \left[\frac{m_i u_i^{(5)} (\sum \sigma_i u_i^{(5)}) - (\sum m_i \sigma_i u_i^{(5)}) u_i^{(5)}}{(\sum \sigma_i u_i^{(5)})^2} \right]_{t-1}. \tag{49}$$

The error to be propagated to the preceding layer is

$$\delta^{(5)}(t) = \frac{\partial r}{\partial a^{(5)}} = \frac{\partial r}{\partial y} = [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1}. \tag{50}$$

Fuzzy similarity measure: In this step, the system will decide if the current structure should be changed or not according to the expected updated amount of the center and width parameters (Eqs. (47) and (49)). To do this, the expected center and width are, respectively, computed as

$$m_{i-\text{new}} = m_i(t) + \Delta m_i(t),$$

$$\sigma_{i-\text{new}} = \sigma_i(t) + \Delta \sigma_i(t). \tag{51}$$

From the current membership functions of output linguistic variables, we want to find the one which is the most similar to the expected membership function by measuring their fuzzy similarity. The fuzzy similarity measure [11] determines the similarity between two fuzzy sets. If A and B are two fuzzy sets with bell-shaped membership functions, then

$$\mu_A(x) = e^{-(x-m_1)^2/\sigma_1^2} \quad \text{and} \quad \mu_B(x) = e^{-(x-m_2)^2/\sigma_2^2}. \tag{52}$$

The approximate fuzzy similarity measure of A and B, $E(A, B)$, is defined and can be computed as follows: Assuming $m_1 \geqslant m_2$,

$$E(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{\sigma_1 \sqrt{\pi} + \sigma_2 \sqrt{\pi} - |A \cap B|}, \tag{53}$$

where $|A \cap B|$ indicates the cardinality of $A \cap B$ and it can be easily computed from

$$|A \cap B| = \frac{1}{2}\frac{h^2(m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2))}{\sqrt{\pi}(\sigma_1 + \sigma_2)} + \frac{1}{2}\frac{h^2(m_2 - m_1 + \sqrt{\pi}(\sigma_1 - \sigma_2))}{\sqrt{\pi}(\sigma_2 - \sigma_1)} + \frac{1}{2}\frac{h^2(m_2 - m_1 - \sqrt{\pi}(\sigma_1 - \sigma_2))}{\sqrt{\pi}(\sigma_1 - \sigma_2)}, \tag{54}$$

where $h(x) = \max\{0, x\}$. The detailed derivation of the fuzzy similarity measure is presented in the appendix.

Let $M(m_i, \sigma_i)$ represent the bell-shaped membership function with center m_i and width σ_i. Let

$$\begin{aligned}\text{degree}(i,t) &= E[M(m_{i-\text{new}}, \sigma_{i-\text{new}}), M(m_{i-\text{closest}}, \sigma_{i-\text{closest}})] \\ &= \max_{1 \le j \le k} E[M(m_{i-\text{new}}, \sigma_{i-\text{new}}), M(m_j, \sigma_j)],\end{aligned} \tag{55}$$

where $k = |T(y)|$ is the size of the fuzzy partition of the output linguistic variable $y(t)$. After the most similar membership function $M(m_{i-\text{closest}}, \sigma_{i-\text{closest}})$ to the expected membership function $M(m_{i-\text{new}}, \sigma_{i-\text{new}})$ has been found, the following adjustment is made:

If degree$(i,t) < \alpha(t)$,
  THEN
    create a new node $M(m_{i-\text{new}}, \sigma_{i-\text{new}})$ in layer 4
      and denote this new node as the *i-closest* node,
    do the structure learning process,
  ELSE IF $M(m_{i-\text{closest}}, \sigma_{i-\text{closest}}) \neq M(m_i, \sigma_i)$
    THEN
      do the structure learning process,
    ELSE
      do the following parameter adjustments in layer 5:

$$m_i(t+1) = m_{i-\text{new}}$$

$$\sigma_i(t+1) = \sigma_{i-\text{new}}$$

      skip the structure learning process.

$\alpha(t)$ is a monotonically increasing scalar similarity index such that *lower similarity* is allowed in the initial stages of learning. According to the above judgement, degree(i,t) is first compared to a given similarity index $\alpha(t)$. If the similarity is too low, then a new term node (new membership function) with the expected parameters is built since, in this case, all the current membership functions are too much different from the expected one. This new node with the expected membership function is created, and the output connections of some just firing rule nodes should be changed to connect to this new term node through the structure learning process. If no new term node is necessary, it will then check if the ith term node is the i-closest node. If this is false, it means that some just firing fuzzy logic rules should have the i-closest (term) node instead of the original ith term node as their consequent. In this case, the structure learning process should be performed to change the current structure properly. If the ith term node is the i-closest node, then no structural change is necessary, and only the parameter learning should be performed by the standard backpropagation algorithm. The structure learning process is then described as follows.

Structure learning process: When entering this process, it means that the ith term node in layer 4 is improperly assigned as the consequent of some fuzzy logic rules which have just been fired *strongly*. The

more proper consequent for these fuzzy logic rules should be the i-closest node. To find the rules whose consequents should be changed, we set a *firing strength threshold* β. Only the rules whose firing strengths are higher than or equal to this threshold are treated as really firing rules. Only the really firing rules are considered to be changing their consequents, since only these rules are fired strongly enough to contribute to the above results of judgement. Assuming that the term node $M(m_i, \sigma_i)$ in layer 4 has inputs from rule nodes $1, \ldots, l$ in layer 3, whose corresponding firing strength are $a_i^{(3)}$'s. $i = 1, \ldots, l$, then

IF $a_i^{(3)}(t) \geqslant \beta$, THEN change the consequent of the ith rule node
from $M(m_i, \sigma_i)$ to $M(m_{i-\text{closest}}, \sigma_{i-\text{closest}})$.

Layer 4: There is no parameter to be adjusted in this layer. Only the error signals ($\delta_i^{(4)}$'s) need to be computed and propagated. The error signal $\delta_i^{(4)}$ is derived as in the following:

$$\delta_i^{(4)} = \frac{\partial r}{\partial a_i^{(4)}} = \frac{\partial r}{\partial u_i^{(5)}} = \frac{\partial r}{\partial a^{(5)}} \frac{\partial a^{(5)}}{\partial u_i^{(5)}}, \tag{56}$$

where from Eq. (9),

$$\frac{\partial a^{(5)}}{\partial u_i^{(5)}} = \frac{m_i \sigma_i (\sum \sigma_i u_i^{(5)}) - (\sum m_i \sigma_i u_i^{(5)}) \sigma_i}{(\sum \sigma_i u_i^{(5)})^2} \tag{57}$$

and from Eq. (50),

$$\frac{\partial r}{\partial a^{(5)}} = \delta^{(5)} = [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1}. \tag{58}$$

Hence, the error signal is

$$\delta_i^{(4)}(t) = [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1} \left[\frac{m_i \sigma_i (\sum \sigma_i u_i^{(5)}) - (\sum m_i \sigma_i u_i^{(5)}) \sigma_i}{(\sum \sigma_i u_i^{(5)})^2} \right]_{t-1}. \tag{59}$$

In the multi-output case, the computations in layers five and four are exactly the same as the above using the same internal reinforcement signals and proceed independently for each output linguistic variable.

Layer 3: As in layer four, only the error signals need to be computed. According to Eq. (7), this error signal can be derived:

$$\delta_i^{(3)}(t) = \frac{\partial r}{\partial a_i^{(3)}} = \frac{\partial r}{\partial u_i^{(4)}} = \frac{\partial r}{\partial a_i^{(4)}} \frac{\partial a_i^{(4)}}{\partial u_i^{(4)}} = \frac{\partial r}{\partial a_i^{(4)}} = \delta_i^{(4)}(t). \tag{60}$$

Hence, the error signal is $\delta_i^{(3)}(t) = \delta_i^{(4)}(t)$. If there is more than one output, then the error signal becomes $\delta_i^{(3)}(t) = \sum_k \delta_k^{(4)}(t)$, 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. (45) and (5), the adaptive rule of m_{ij} is derived:

$$\frac{\partial r}{\partial m_{ij}} = \frac{\partial r}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial m_{ij}} = \frac{\partial r}{\partial a_i^{(2)}} e^{f_i} \frac{2(u_i^{(2)} - m_{ij})}{\sigma_{ij}^2}, \tag{61}$$

where from Eq. (60),

$$\frac{\partial r}{\partial a_i^{(2)}} = \sum_i \frac{\partial r}{\partial a_i^{(3)}} \frac{\partial a_i^{(3)}}{\partial u_i^{(3)}}, \tag{62}$$

$$\frac{\partial r}{\partial a_i^{(3)}} = \delta_i^{(3)}, \tag{63}$$

and from Eq. (6),

$$\frac{\partial a_i^{(3)}}{\partial u_i^{(3)}} = \begin{cases} 1 & \text{if } u_i^{(3)} = \min(\text{inputs of rule node } k), \\ 0 & \text{otherwise.} \end{cases} \tag{64}$$

Hence,

$$\frac{\partial r}{\partial a_i^{(2)}} = \sum_k q_k(t), \tag{65}$$

where the summation is performed over the rule nodes that $a_i^{(2)}$ feeds into, and

$$q_k(t) = \begin{cases} \delta_k^3(t) & \text{if } a_i \text{ is minimum in the } k\text{th rule node's input,} \\ 0 & \text{otherwise.} \end{cases} \tag{66}$$

So the adaptive rule of m_{ij} is

$$m_{ij}(t+1) = m_{ij}(t) + \eta \left[\frac{\partial r}{\partial a_i^{(2)}}\right]_t \left[e^{f_i} \frac{2(u_i^{(2)} - m_{ij})}{\sigma_{ij}^2}\right]_{t-1}. \tag{67}$$

Similarly, using Eqs. (45), (5), and (62)–(66), the adaptive rule of σ_{ij} is derived:

$$\frac{\partial r}{\partial \sigma_{ij}} = \frac{\partial r}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial \sigma_{ij}} = \left[\frac{\partial r}{\partial a_i^{(2)}}\right]_t \left[e^{f_i} \frac{2(u_i^{(2)} - m_{ij})^2}{\sigma_{ij}^3}\right]_{t-1}. \tag{68}$$

Hence, the adaptive rule of σ_{ij} becomes

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) + \eta \left[\frac{\partial r}{\partial a_i^{(2)}}\right]_t \left[e^{f_i} \frac{2(u_i^{(2)} - m_{ij})^2}{\sigma_{ij}^3}\right]_{t-1}. \tag{69}$$

Note that we perform structure learning only in the output part of the RNFCN (for the output linguistic variables) in the above learning algorithm. In fact, the same structure learning scheme can be applied to the input part of the RNFCN for the input linguistic variables, since the error signals for layer-two nodes are available in Eqs. (67) and (69). This can possibly automate the choice of the number of input fuzzy partition. However, simulation results show that the error values are usually too small to perform structure changes in this layer due to the nature of the backpropagation algorithm. Other structure learning schemes need to be developed to address this problem.

4.2. Fuzzy predictor

We shall use an NFCN to develop a fuzzy predictor (evaluation network) as shown in Fig. 4. It shares the same fuzzifier as the action network; that is, both use the same internal representation, which is an overlapping type of distributed representation of input patterns. The fuzzy predictor receives an external reinforcement signal from the environment and produces internal reinforcement signals to the action network. There are two kinds of fuzzy predictors: single-step fuzzy predictor and multi-step fuzzy predictor, suitable to different reinforcement learning problems. The function of the single-step fuzzy predictor is to predict the external reinforcement signal $r(t)$ one time step ahead, that is, at time $t-1$. Here, $r(t)$ is the real reinforcement signal resulting from the inputs and actions chosen at time step $t-1$, but it can only be known at time step t. If the fuzzy predictor can produce a signal $p(t)$, which is the prediction of $r(t)$ but is available at time step $t-1$, then the time delay problem can be solved. With a correct predicted signal $p(t)$, a better action

can be chosen by the action network at time step $t-1$, and the corresponding learning can be performed on the action network at time step t upon receiving the external reinforcement signal $r(t)$. As indicated in the last subsection, $p(t)$ is necessary for the stochastic exploration with multi-parameter probability distribution (Eq. (39)). The other internal reinforcement signal $\hat{r}(t)$ in Fig. 4 is set as $\hat{r}(t) = r(t) - p(t)$, which is the prediction error for computing Eq. (42) by the action network. Basically, the training of a single-step predictor is a simple supervised learning problem. Thus, the reinforcement learning algorithm for the single-step fuzzy predictor is exactly the same as the on-line supervised learning algorithm proposed for the NFCN with a single output node. The single-step prediction is the extreme case of the multi-step prediction. Hence, we will focus on the multi-step fuzzy predictor in the following.

When both the reinforcement signal and input patterns from the environment may depend arbitrarily on the past history of the network output and the network may only receive a reinforcement signal after a long sequence of outputs, the credit assignment problem becomes severe. This *temporal credit assignment* problem results because we need to assign credit or blame to each step individually in such a long sequence for an eventual success or failure. Hence, for this class of reinforcement learning problem, we need to solve the temporal credit assignment problem together with the original structure credit assignment problem of attributing network error to different connections or weights. The solution to the temporal credit assignment problem in the RNFCN is to design a multi-step fuzzy predictor which can predict the reinforcement signal at each time step within two successive external reinforcement signals which may be separated by many time steps. This multi-step fuzzy predictor can assure that both the evaluation network and the action network do not have to wait until the actual outcome is known, and they can update their parameters and structures within the period without any evaluative feedback from the environment. To solve the temporal credit assignment problem, the technique based on the temporal-difference methods is used [23]. Unlike the single-step prediction or the supervised learning method which assigns credit according to the difference between the predicted and actual output, the temporal-difference methods assign credit according to the difference between temporally successive predictions. Some important temporal-difference equations of three different cases are summarized below.

Case 1: *Prediction of final outcome.* Given the observation-outcome sequences of the form $x_1, x_2, \ldots, x_m, z$, where each x_t is an input vector available at time step t from the environment, and z is the external reinforcement signal available at time step $m+1$. For each observation-outcome sequence, the fuzzy predictor produces a corresponding sequence of predictions $p_1, p_2, \ldots, p_m$, each of which is an estimate of z. Since p_t is the output of the evaluation network at time t, p_t is a function of the network's input x_t, and the network's adjustable parameters w_t, and can be denoted as $p(x_t, w_t)$, where w_t can be $m_i(t)$ (center of membership function) or $\sigma_i(t)$ (width of membership function). For this prediction problem, the learning rule, which is called TD(λ) family of learning procedures, is

$$\Delta w_t = \eta(p_t - p_{t-1}) \sum_{k=1}^{t-1} \lambda^{t-k-1} \nabla_w p_k, \tag{70}$$

where $p_{m+1} \equiv z$, $0 \leqslant \lambda \leqslant 1$, and η is the learning rate. λ is the recency weighting factor with which alternations to the predictions of observation vectors occurring k steps in the past are weighted by λ^k. In the extreme case that $\lambda = 1$, all the proceeding predictions, $p_1, p_2, \ldots, p_{t-1}$, are altered properly according to the current temporal difference, $p_t - p_{t-1}$, to an "equal" extent. In this case, Eq. (70) reduces to a supervised-learning approach, and if p_t is a linear function of x_t and w_t, then it is the same as the Widrow–Hoff procedure [6]. In the other extreme case that $\lambda = 0$, the increment of the parameter w, is determined only by its effect on the prediction associated with the most recent observation. A theorem about the convergence of TD(0) when p_t is a linear function of x_t and w_t can be found in [23].

Case 2: *Prediction of finite cumulative outcomes.* In this case, p_t predicts the remaining cumulative cost given the tth observation x_t, rather than the overall cost for the sequence. This case happens when we are

more concerned with the sum of future predictions than the prediction of what will happen at a specific future time. Let r_t be the actual cost incurred between time steps $t-1$ and t. Then p_{t-1} is to predict $z_{t-1} = \sum_{k=t}^{m+1} r_k$. Hence, the prediction error is

$$z_{t-1} - p_{t-1} = \sum_{k=1}^{m+1} r_k - p_{t-1} = \sum_{k=1}^{m+1} (r_k + p_k - p_{k-1}),$$

where p_{m+1} is defined as 0. Thus, the learning rule is

$$\Delta w_t = \eta(r_t + p_t - p_{t-1}) \sum_{k=1}^{t-1} \lambda^{t-k-1} \nabla_w p_k. \tag{71}$$

Case 3: *Prediction of infinite discounted cumulative outcomes.* In this case, p_{t-1} predicts $z_{t-1} = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = r_t + \gamma p_t$, where the discount-rate parameter γ, $0 \leqslant \gamma < 1$, determines the extent to which we are concerned with short- or long-range prediction. This is used for prediction problems in which exact success or failure may never become completely known. In this case, the prediction error is $(r_t + \gamma p_t) - p_{t-1}$, and the learning rule is

$$\Delta w_t = \eta(r_t + \gamma p_t - p_{t-1}) \sum_{k=1}^{t-1} \lambda^{t-k-1} \nabla_w p_k. \tag{72}$$

In applying the temporal difference procedures to the proposed RNFCN, we let $\lambda = 0$ due to its efficiency and accuracy [23]. A general learning rule used for the above three cases is

$$\Delta w_t = \eta(r_t + \gamma p_t - p_{t-1}) \nabla_w p_{t-1}, \tag{73}$$

where γ, $0 \leqslant \gamma < 1$, is a discount-rate parameter, and η is the learning rate.

We shall next derive the learning rule of the multi-step fuzzy predictor according to Eq. (73). In this case, $p(t)$ is the single output of the fuzzy predictor (evaluation network) for the network's current parameter $w(t)$, and current given input vector $x(t)$, at time step t. Here, $p(t)$ can be any kind of prediction output in the various cases of the multi-step prediction problem stated above. According to Eq. (73), let

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1), \quad 0 \leqslant \gamma < 1. \tag{74}$$

Then $\hat{r}(t)$ is the error signal of the output node of the multi-step fuzzy predictor. The general parameter learning rule then is

$$\Delta w(t) = \eta \hat{r}(t) \left[\frac{\partial p}{\partial w} \right]_{t-1}, \tag{75}$$

where w is the network parameter (i.e., m_i or σ_i). The learning rule for each layer in the fuzzy predictor can be computed as in Eqs. (45)–(69). The only exception is that the error signal is different. Thus, the learning equations for the multi-step fuzzy predictor are the same as in Eqs. (45)–(69) but with the term

$$[r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma} \right]_{t-1}$$

replaced by the term $\hat{r}(t)$ in Eq. (74). Also the multi-step fuzzy predictor will provide two internal reinforcement signals, the prediction output $p(t)$, and the prediction error $\hat{r}(t)$, to the action network for its learning (see Fig. 5).

The learning algorithm for the action network is the same as that derived in Section 4.1 above. However, due to the different nature of the internal reinforcement signal $\hat{r}(t)$, the learning algorithm of the action network with the multi-step fuzzy predictor will be different. The goal of the action network is to maximize

the external reinforcement signal $r(t)$. Thus, we need to estimate the gradient information $\partial r/\partial y$ as we did above. With the internal reinforcement signals $p(t)$ and $\hat{r}(t)$, from the evaluation network, the action network can perform the stochastic exploration and learning. The prediction signal $p(t)$ is used to decide the variance of the normal distribution function in the stochastic exploration in Eq. (39). Then the actual output $\hat{y}(t)$ can be determined according to Eq. (40). Since $\hat{r}(t)$ is the prediction error, the gradient information is estimated as

$$\frac{\partial r}{\partial y} = \hat{r}(t)\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}. \tag{76}$$

In Eq. (74), the prediction error is $\hat{r}(t) = r(t) + \gamma p(t) - p(t-1) = r(t) - [p(t-1) - \gamma p(t)]$. Since $p(t-1)$ predicts the accumulated reinforcement signal in the future (i.e. $r(t) + \gamma p(t)$), $p(t-1) - \gamma p(t)$ predicts the next reinforcement signal (i.e., $r(t)$). Thus, $r(t)$ is the reinforcement signal with respect to the actual action $\hat{y}(t-1)$, and $[p(t-1) - \gamma p(t)]$ is the reinforcement signal with respect to the expected action $y(t-1)$. Then from the equation

$$\frac{\partial r}{\partial y} = [r(t) - [p(t-1) - \gamma p(t)]]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}, \tag{77}$$

we can observe that if $r(t) > [p(t-1) - \gamma p(t)]$, the actual action $\hat{y}(t-1)$ is better than the expected action $y(t-1)$. So $y(t-1)$ should be moved closer to $\hat{y}(t-1)$. On the other side, if $r(t) < [p(t-1) - \gamma p(t)]$, then the actual action $\hat{y}(t-1)$ is worse than the expected action $y(t-1)$. So $y(t-1)$ should be moved further away from $\hat{y}(t-1)$.

Having the gradient information $\partial r/\partial y$ (Eq. (77)), the learning algorithm of the action network can be determined in the same way as in the previous section. The exact learning equations are the same as in Eqs. (43)–(69) except that

$$[r(t) - p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}$$

has been replaced by the new error term

$$[r(t) + \gamma p(t) - p(t-1)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}.$$

5. Illustrative examples

A general purpose simulator has been written in the "C" language to simulate and show the applicability of the proposed systems. Using this simulator, two typical examples are presented in this section. The first example is to use NFCN to control an unmanned vehicle by learning the driving technique of a skilled driver, and the second example is to use RNFCN to solve the cart–pole balancing problem.

5.1. Example 1: fuzzy control of unmanned vehicle

This example illustrates the applicability of the proposed two-phase hybrid learning algorithm for constructing the NFCN for the control of the "fuzzy" car conceived by Sugeno [21]. The car has the ability to learn from training examples to move automatically along a track with rectangular turns. The goal is to demonstrate that the car, under a NFCN, can learn from past driving experiences of a skilled driver, and then the car can run automatically for similar road conditions as if it were driven by a skilled driver. The input linguistic variables are x_0, x_1, and x_2 which represent the distance of the car from the side boundary of the

track, the distance of the car from the turning point of a corner, and the current steering angle, respectively (see Fig. 6). The output linguistic variable y is the next steering angle. The constraints of these variables are $0 \leqslant x_0 \leqslant 250$ cm, $0 \leqslant x_1 \leqslant 700$ cm, and $-65° \leqslant x_2, y \leqslant 65°$. The training data are obtained in the process when a skilled operator guides the fuzzy car along the track as shown in Fig. 9. A total of 780 input–output training pairs are used. In the simulation, we set the side of fuzzy partitions of x_0, x_1, and x_2 to 3, 5, and 5, respectively; that is, the input linguistic variable x_0 has three fuzzy sets ("close", "normal", and "far") in describing the distance of the car from the side of the track.

In the simulation, the two-phase hybrid learning algorithm is used to set up the proposed NFCN with sets of off-line training data. In this simulation, the size of fuzzy partitions of the output linguistic variable is set to 10 and the overlap parameter is set as $r = 2.0$. Figure 7 shows the curve of mean-squared error with respect to the number of epochs, and the curve of mean iteration number with respect to the number of epochs. Here the learning rate is set to 0.15 and the error tolerance is 0.01. From the second curve, we can find that the average iteration number for a single training point to converge is rather small from the beginning, meaning that the phase-one, self-organizing learning process has done much work already. The learned NFCN is shown in Fig. 8. After the whole connectionist fuzzy logic controller is established, it is used to control the car. We keep the speed of the car constant, and assume that there are sensors on the car to measure the state values x_0, x_1, and x_2 which are fed into the controller to derive the next steering angle. The simulated result is shown in

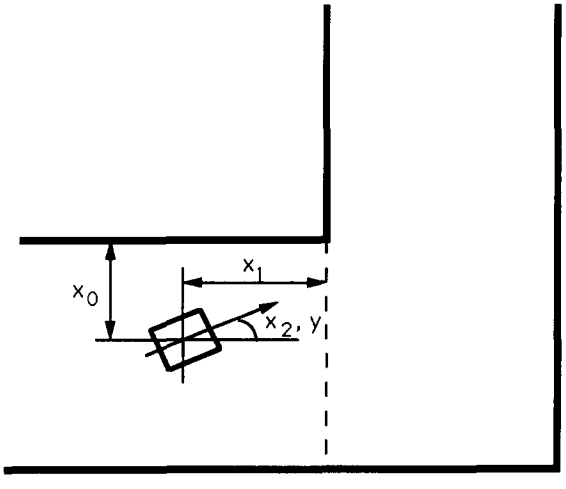

Fig. 6. The state variables in the fuzzy car example.

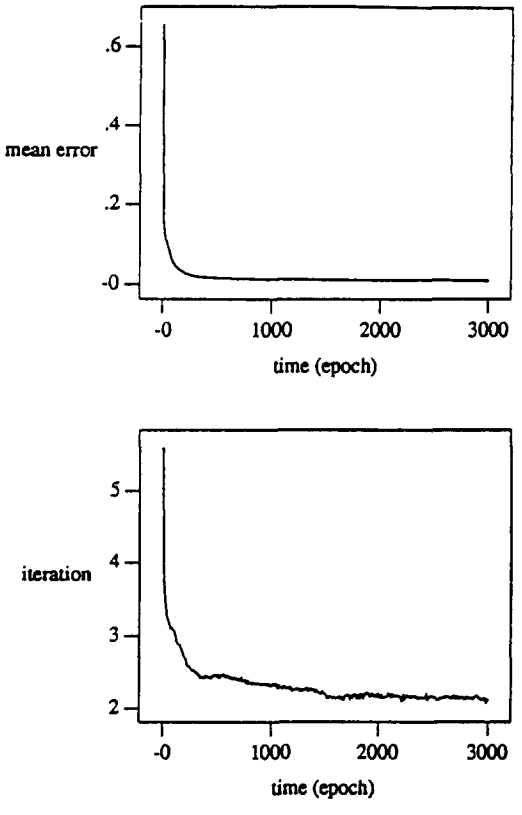

Fig. 7. Learning curves: mean error and mean iteration versus time (epoch) in the fuzzy car simulation using hybrid learning algorithm.

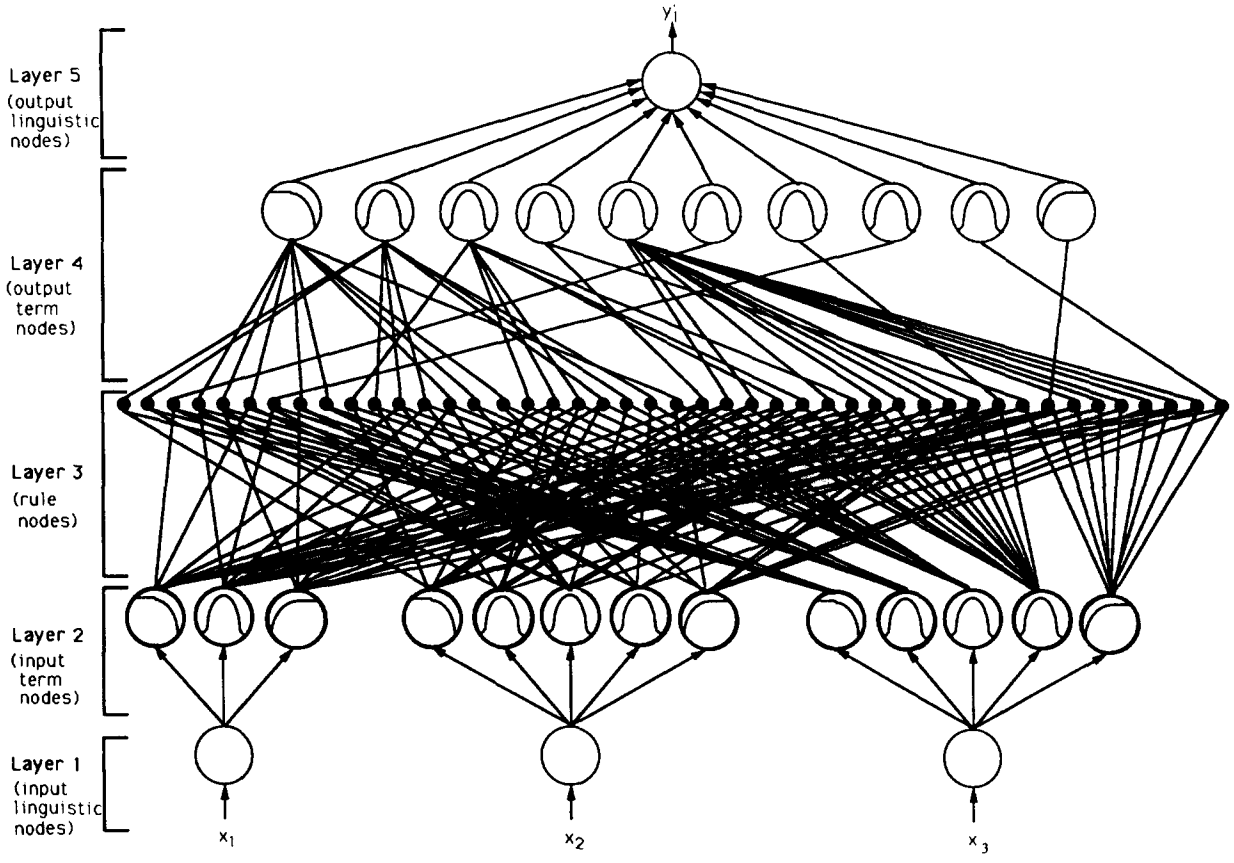

Fig. 8. Structure of the NFCN after the supervised learning in the fuzzy car simulation.

Fig. 9. The dotted curve is the training path and the solid curve is the path that the fuzzy car runs under the control of the proposed connectionist fuzzy logic controller. We found that these paths coincide closely. We simulated this example several times with different initial steering angles, and the results we obtained were nearly the same.

5.2. *Example 2: the cart–pole balancing problem*

The proposed RNFCN with the multi-step fuzzy predictor has been simulated for the cart–pole balancing problem or the so-called inverted pendulum balancing problem. This problem is often used as an example of inherently unstable, dynamic systems to demonstrate both modern and classic control techniques as well as the learning control techniques of neural networks using supervised learning methods [28] or reinforcement learning methods [1].

As shown in Fig. 10, the cart–pole balancing problem is the problem of learning how to balance an upright pole. The bottom of the pole is hinged to a cart that travels along a finite-length track to its right or its left. Both the cart and the pole can move only in the vertical plane; that is, each has only one degree of freedom. There are four input state variables in this system: θ, angle of the pole from an upright position (in degrees); $\dot{\theta}$, angular velocity of the pole (in degrees/s); x, horizontal position of the cart's center (in meters); and $\dot{x}$, velocity

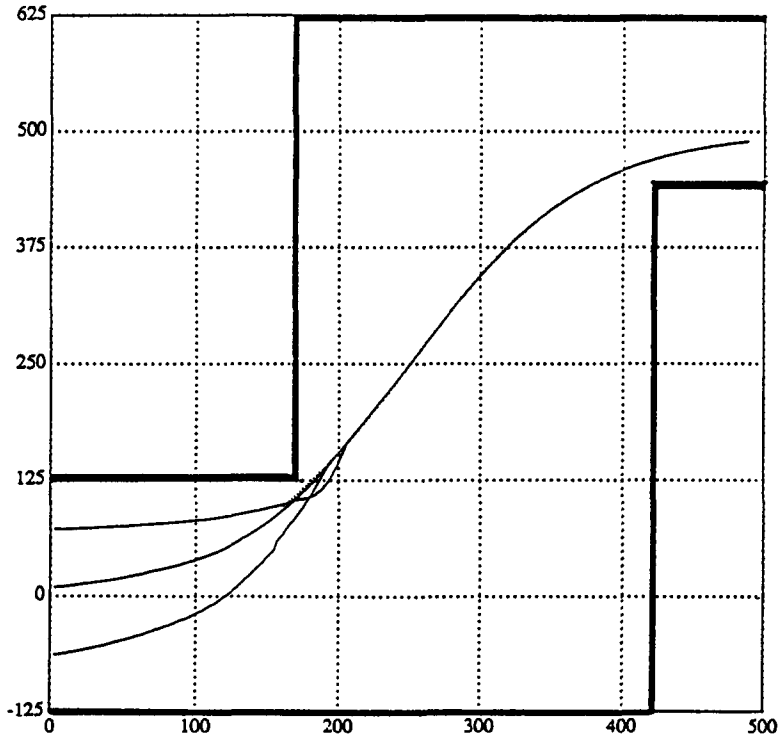

Fig. 9. Simulation results of the fuzzy car running under the control of the learned NFCN in the fuzzy car simulation, where the dotted line is the training curve and solid lines are the running curves of the fuzzy cars.

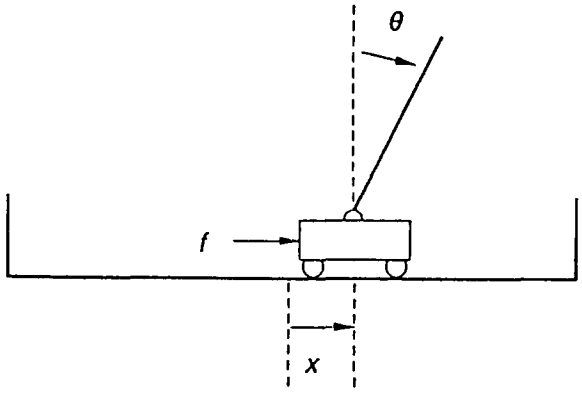

Fig. 10. The cart–pole balancing system.

of the cart (in m/s). The only control action is f, which is the amount of force (N) applied to the cart to move it toward its left or right. The system fails and receives a penalty signal of -1 when the pole falls past a certain angle ($\pm 12°$ is used here) or the cart runs into the bounds of its track (the distance is 2.4 m from the center to both bounds of the track). The goal of this control problem is to train the RNFCN such that it can determine

a sequence of forces with proper magnitudes to apply to the cart to balance the pole for as long as possible without failure.

The model and the corresponding parameters of the cart–pole balancing system for our computer simulation are adopted from [1] with additional consideration of friction effects. This model and its parameters are also used by Barto et al. [1]. The equations of motion that we used are

$$\theta(t+1) = \theta(t) + \Delta\dot{\theta}(t), \tag{78}$$

$$\dot{\theta}(t+1) = \dot{\theta}(t) + \Delta \frac{mg \sin\theta(t) - \cos\theta(t)[f(t) + m_p l(\dot{\theta}(t)\pi/180)^2 \sin\theta(t) - \mu_c \operatorname{sgn}(\dot{x}(t))] - \mu_p m\dot{\theta}(t)/m_p l}{(4/3)ml - m_p l\cos^2\theta(t)},$$

$$x(t+1) = x(t) + \Delta\dot{x}(t),$$

$$\dot{x}(t+1) = \dot{x}(t) + \Delta \frac{f(t) + m_p l[(\dot{\theta}(t)\pi/180)^2 \sin\theta(t) - \ddot{\theta}(t)\pi/180\cos\theta(t)] - \mu_c \operatorname{sgn}[\dot{x}(t)]}{m},$$

where $g = -9.8\,\mathrm{m/s^2}$, acceleration due to the gravity, $m = 1.1\,\mathrm{kg}$, combined mass of the pole and the cart, $m_p = 0.1\,\mathrm{kg}$, mass of the pole, $l = 0.5\,\mathrm{m}$, half-pole length, $\mu_c = 0.0005$, coefficient of friction of the cart on the track, $\mu_p = 0.000002$, coefficient of friction of the pole on the cart, and $\Delta = 0.02$, sampling interval.

The constraints on the variables are $-12° \leqslant \theta \leqslant 12°$, $-2.4\,\mathrm{m} \leqslant x \leqslant 2.4\,\mathrm{m}$, and $-10\,\mathrm{N} \leqslant f \leqslant 10\,\mathrm{N}$. In designing the controller, the equations of motion of the cart–pole balancing system are assumed to be *unknown* to the controller. A more challenging part of this problem is that the only available feedback is a failure signal that notifies the controller when a failure occurs; that is, either $|\theta| > 12°$ or $|x| > 2.4\,\mathrm{m}$. This is a typical reinformation learning problem and the feedback failure signal serves as the reinforcement signal. Since a reinforcement signal may only be available after a long sequence of time steps in this failure avoidance task, a multi-step fuzzy predictor is required for the RNFCN. Moreover, since the goal is to avoid failure for as long as possible, there is no exact success in finite time. Also, we hope that the RNFCN can balance the pole for as long as possible for infinite trials, not just for one particular trial, where a "trial" is defined as the time steps from an initial state to a failure. Hence, Eq. (73) is used here for the temporal-difference prediction method. The reinforcement signal is defined as

$$r(t) = \begin{cases} -1 & \text{if } |\theta(t)| > 12° \text{ or } |x(t)| > 2.4\,\mathrm{m}, \\ 0 & \text{otherwise}, \end{cases} \tag{79}$$

and the goal is to maximize the sum $\sum_{k=0}^{\infty} \gamma^k r(t+k)$, where γ is the discount rate.

In our computer simulation, the learning system was tested for 10 runs by trying to use the same learning parameter values in [1]. Each run consisted of a sequence of trials; each trial began with the same initial condition $\theta(0) = \dot{\theta}(0) = x(0) = \dot{x}(0) = 0$, or with a randomized initial condition, and ended with a failure signal indicating that either $|\theta(t)| > 12°$ or $|x(t)| > 2.4\,\mathrm{m}$. The randomized initial condition means that after each failure, the initial configuration was independently and randomly chosen such that $-10 < \theta(0) < 10$, $-50 < \dot{\theta}(0) < 50$, $-2 < x(0) < 2$, and $-10 < \dot{x}(0) < 10$. The input fuzzy partitions were set as $|T(x)| = 3$, $|T(\dot{x})| = 3$, $|T(\theta)| = 7$, and $|T(\dot{\theta})| = 3$ for all runs. For each run, the input (output) membership functions were initialized so that they covered the whole input (output) space evenly, and the output fuzzy partition was initialized as $|T(f)| = 7$. The membership functions were chosen to be the bell-shaped functions (Eq. (5)). Also, in the initiation of each run, each rule was assigned with a consequent term randomly. There is a total of 189 rules in the beginning. Here we assumed that no expert knowledge (in the form of IF-THEN rules) are available. Runs were consisted of at most 50 trials unless the duration of each

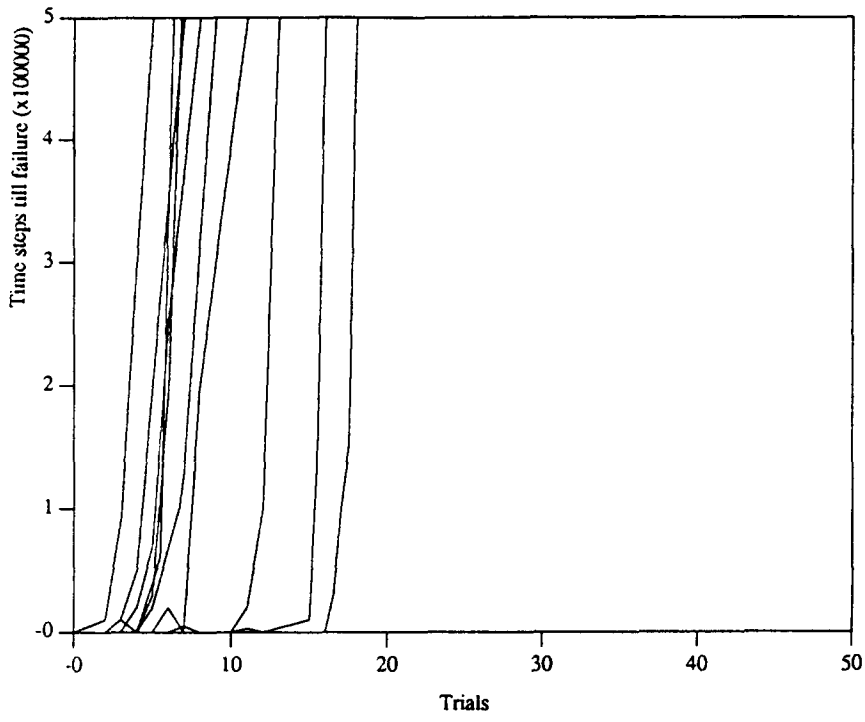

Fig. 11. Performance of the RNFCN on the cart–pole balancing problem.

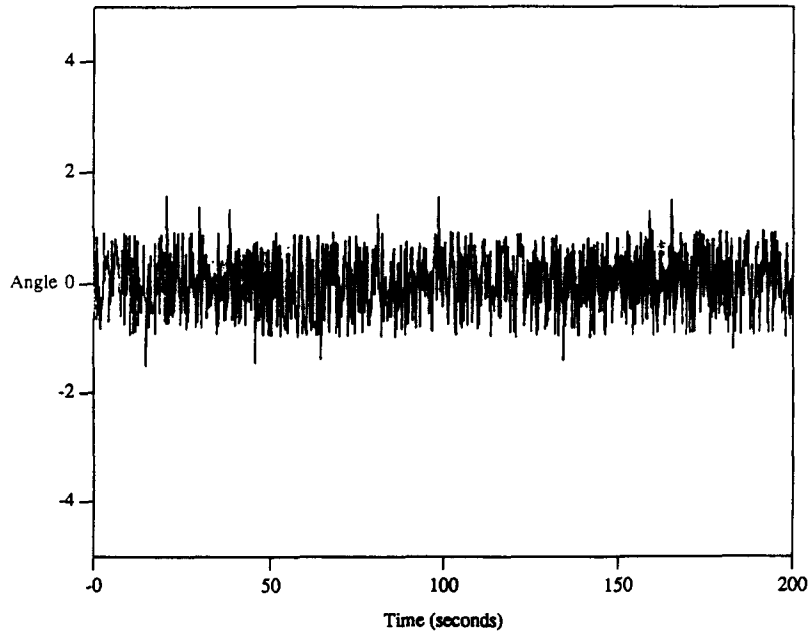

Fig. 12. Variations in θ produced by the learned RNFCN.

run exceeded 500 000 time steps. A run was successful and terminated after 500 000 time steps before all 50 trials took place; otherwise, it was called "a failure" and terminated at the end of its 50th trial.

In our computer simulations, a total of 10 runs were performed. Among these 10 runs, five of them started with zero initial conditions and the others started with randomized initial conditions. The simulation results (see Fig. 11) showed that the RNFCN can learn to balance the pole within 20 trials. In most of the 10 runs, the learning was completed before 10 trials. It was observed that the runs starting with randomized initial condition usually took more trials. Fig. 12 shows the angle deviation of the pole about the center point when the cart–pole system was controlled by a well-trained RNFCN. This performance is better than the results presented in [1] and compatible to those in [3]. In most runs, the final number of learned output membership functions is less than 15 as compared to 189 output membership functions that were used in [9] for each run; that is, one output membership function for each (overlapping) grid of input space.

6. Conclusion

A general connectionist model of a fuzzy logic control system called NFCN was proposed. The proposed model introduces the low-level learning power of neural networks into the fuzzy logic system and provides high-level human-understandable meaning to the normal connectionist architecture. A hybrid learning scheme which combines a self-organized learning algorithm and a supervised learning algorithm was developed for performing the structure and parameter learning of this model. The hybrid learning algorithm performed well when sets of precise supervised training data are available. We also described the development of integrating two NFCNs into an integrated RNFCN for solving various reinforcement learning problems. By combining the techniques of temporal difference, stochastic exploration, and a derived on-line supervised structure–parameter learning algorithm, a reinforcement structure–parameter learning algorithm was proposed for the RNFCN. The proposed RNFCN makes the design of fuzzy logic controllers more practical for real-world applications since it greatly lessens the quality and quantity requirements of the feedback training signals. Computer simulations of the unmanned vehicle control and the cart–pole balancing problem satisfactorily verified the validity and the performance of the proposed supervised learning algorithm for NFCN and the reinforcement structure–parameter learning algorithm for the RNFCN, respectively.

Appendix: approximate similarity measure of fuzzy sets

We use an approximate approach to reduce the computational complexity of the fuzzy similarity measure of two fuzzy sets with bell-shaped membership functions. Since the area of the bell-shaped function $e^{-(x-c)^2/\sigma^2}$ is $\sigma\sqrt{\pi}$ and its height is always 1, we can approximate it by an isosceles triangle with unity height and the length of bottom edge $2\sigma\sqrt{\pi}$. We can then compute the fuzzy similarity measure of two fuzzy sets with such kind of membership functions (see Fig. 13).

To derive the equations of the similarity measure of two fuzzy sets with isosceles triangular membership functions, we observe all the possible relative relationships of two isosceles triangles on a horizontal axis (x-axis). Let $\Delta(m_i, \sigma_i)$ denote the isosceles triangle with unity height, bottom-line length $2\sigma_i\sqrt{\pi}$, and center of bottom-line at m_i on x-axis. Assume the two end-points of the bottom-line of $\Delta(m_1, \sigma_1)$ are a and b on x-axis, and the two end-points of the bottom-line of $\Delta(m_2, \sigma_2)$ are c and d on x-axis (see Fig. 13). That is,

$$a = m_1 - \sigma_1\sqrt{\pi}, \quad b = m_1 + \sigma_1\sqrt{\pi}, \quad c = m_2 - \sigma_2\sqrt{\pi}, \quad d = m_2 + \sigma_2\sqrt{\pi}.$$

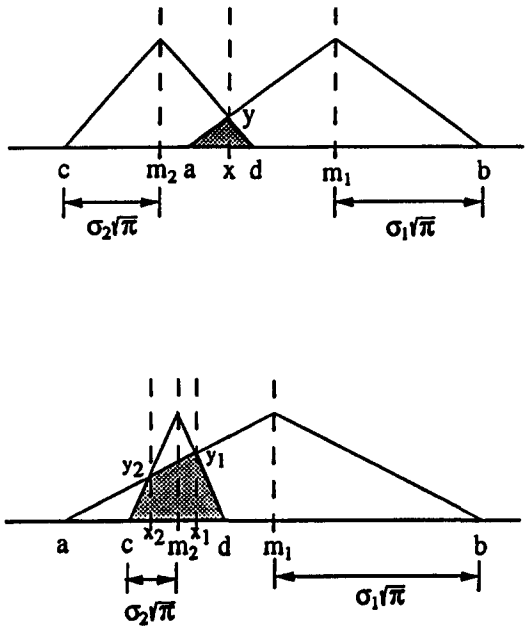

Fig. 13. Two isosceles triangular membership functions.

First, if $m_1 = m_2$, then obviously, $|A \cap B| = \sigma_2\sqrt{\pi}$ (if $\sigma_1 \geqslant \sigma_2$) or $|A \cap B| = \sigma_1\sqrt{\pi}$ (if $\sigma_1 < \sigma_2$). In the following discussion, we assume $m_1 > m_2$. Let us consider the following four possible situations:

Case 1: If $a \geqslant d$, then $|A \cap B| = 0$ since the two membership functions do not overlap.

Case 2: If $b \geqslant d > a \geqslant c$, then

$$|A \cap B| = \tfrac{1}{2}(d-a)y$$

$$= \frac{1}{2}\frac{(m_2 - m_1 + \sigma_1\sqrt{\pi} + \sigma_2\sqrt{\pi})^2}{(\sigma_1 + \sigma_2)\sqrt{\pi}}. \tag{A.1}$$

Case 3: If $b > d$ and $c > a$, then

$$|A \cap B| = \tfrac{1}{2}(x_2 - c)y_2 + \tfrac{1}{2}(y_1 + y_2)(x_1 - x_2) + \tfrac{1}{2}(d - x_1)y_1. \tag{A.2}$$

We can get

$$\begin{aligned} \frac{1}{2}(x_2 - c)y_2 &= \frac{1}{2}\sigma_2\frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 - \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 - \sigma_2)^2}, \\ \frac{1}{2}(d - x_1)y_1 &= \frac{1}{2}\sigma_2\frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 + \sigma_2)^2}. \end{aligned} \tag{A.3}$$

Since

$$\tfrac{1}{2}(y_1 + y_2)(x_1 - x_2) = \tfrac{1}{2}(x_1 - a)y_1 - \tfrac{1}{2}(x_2 - a)y_2. \tag{A.4}$$

we can get the following results:

$$\begin{aligned}\frac{1}{2}(x_1 - a)y_1 &= \frac{1}{2}\sigma_1 \frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 + \sigma_2)^2},\\ \frac{1}{2}(x_2 - a)y_2 &= \frac{1}{2}\sigma_1 \frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 - \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 - \sigma_2)^2}.\end{aligned} \tag{A.5}$$

So the final result is that

$$|A \cap B| = \frac{1}{2}\frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 - \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 - \sigma_2)} + \frac{1}{2}\frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 + \sigma_2)}. \tag{A.6}$$

Case 4: If $b < d$ and $a > c$, then

$$|A \cap B| = \tfrac{1}{2}(b - x_1)y_1 + \tfrac{1}{2}(x_2 - a)y_2 + \tfrac{1}{2}(d - x_2)y_2 - \tfrac{1}{2}(d - x_1)y_1. \tag{A.7}$$

We can get the following results:

$$\begin{aligned}\frac{1}{2}(b - x_1)y_1 &= \frac{1}{2}\sigma_1 \frac{[m_1 - m_2 + \sqrt{\pi}(\sigma_1 - \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 - \sigma_2)^2},\\ \frac{1}{2}(x_2 - a)y_2 &= \frac{1}{2}\sigma_1 \frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 + \sigma_2)^2},\\ \frac{1}{2}(d - x_2)y_2 &= \frac{1}{2}\sigma_2 \frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 + \sigma_2)^2},\\ \frac{1}{2}(d - x_1)y_1 &= \frac{1}{2}\sigma_2 \frac{[m_1 - m_2 + \sqrt{\pi}(\sigma_1 - \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 - \sigma_2)^2}.\end{aligned} \tag{A.8}$$

So, we can get

$$|A \cap B| = \frac{1}{2}\frac{[m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 + \sigma_2)} + \frac{1}{2}\frac{[m_1 - m_2 + \sqrt{\pi}(\sigma_1 - \sigma_2)]^2}{\sqrt{\pi}(\sigma_1 - \sigma_2)}. \tag{A.9}$$

From the above discussion and the truth that $m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2) < 0$ implies $m_2 - m_1 + \sqrt{\pi}(\sigma_1 - \sigma_2) < 0$ implies $m_2 - m_1 + \sqrt{\pi}(\sigma_2 + \sigma_1) < 0$, we can conclude a general formula for $|A \cap B|$ as in Eq. (54). Notice that this general formula is true even when $m_1 = m_2$.

References

[1] A.G. Barto, R.S. Sutton and C.W. Anderson, Neuronlike adaptive elements that can solve difficult learning control problems, *IEEE Trans. System Man Cybernet.* **SMC-13** (1983) 834–847.

[2] H.R. Berenji, An architecture for designing fuzzy controllers using neural networks, *Internat. J. Approx. Reasoning* **6** (1992) 267–292.

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

[4] H. Bersini, J.-P. Nordvik and A. Bonarini, A simple direct adaptive fuzzy controller derived from its neural equivalent, *Proc. IEEE Internat. Conf. on Neural Networks*, San Francisco, CA (1993) 345–350.

[5] I. Enbutsu, K. Baba and N. Hara, Fuzzy rule extraction from a multilayered neural network, *Proc. IEEE Internat. Joint Conf. on Neural Networks* (1991) II-461–465.

[6] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation* (Addison-Wesley, New York, 1991) 188–189.
[7] J.-S. Jang, Self-learning fuzzy controllers based on temporal back propagation, *IEEE Trans. Neural Networks* **3** (1992) 741–723.
[8] C.C. Lee, Fuzzy logic in control systems: fuzzy logic controller – part I & II, *IEEE Trans. System Man Cybernet.* **SMC-20** (1990) 404–435.
[9] C.C. Lee and H.R. Berenji, An intelligent controller based on approximate reasoning and reinforcement learning, *Proc. IEEE Intelligent Machine* (1989) 200–205.
[10] C.T. Lin and C.S.G. Lee, Neural-network-based fuzzy logic control and decision system, *IEEE Trans. Comput.* **C-40** (1991) 1320–1336.
[11] C.T. Lin and C.S.G. Lee, Real-time supervised structure–parameter learning for fuzzy neural network, *Proc. IEEE Internat. Conf. on Fuzzy Systems*, San Diego, CA (1992) 1283–1290.
[12] C.T. Lin and C.S.G. Lee, Reinforcement structure-parameter learning for neural-network-based fuzzy logic control systems, *Proc. IEEE Internat. Conf. on Fuzzy Systems*, San Francisco, CA (1993) 88–93.
[13] E. Khan and P. Venkatapuram, Neufuz: neural network based fuzzy logic design algorithms, *Proc. IEEE Internat. Conf. on Fuzzy Systems*, San Francisco, CA (1993) 647–654.
[14] T. Kohonen, *Self-organization and Associative Memory* (Springer, Berlin, 1988) p. 132.
[15] B. Kosko, *Neural Networks and Fuzzy Systems* (Prentice-Hall, Englewood Cliffs, NJ, 1992).
[16] A. Nafarieh and J.M. Keller, A new approach to inference in approximate reasoning, *Fuzzy Sets and Systems* **41** (1991) 17–37.
[17] K.S. Narendra and M.A.L. Thathachar, *Learning Automata: An Introduction* (Prentice-Hall, Englewood Cliffs, NJ, 1989).
[18] D. Nauck and R. Kruse, A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backpropagation, *Proc. IEEE Internat. Conf. on Neural Networks*, San Francisco, CA (1993) 1022–1027.
[19] H. Nomura, I. Hayashi and N. Wakami, A learning method of fuzzy inference rules by descent method, *Proc. IEEE Internat. Conf. on Neural Networks* (1992) 203–210.
[20] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, in: *Parallel Distributed Processing*, Vol. 1 (MIT Press, Cambridge, 1986) 318–362.
[21] M. Sugeno and M. Nishida, Fuzzy control of model car, *Fuzzy Sets and Systems* **16** (1985) 103–113.
[22] C.-T. Sun and J.-S. Jang, A neuro-fuzzy classifier and its applications, *Proc. IEEE Internat. Conf. on Fuzzy Systems*, San Francisco, CA (1993) 94–98.
[23] R.S. Sutton, Learning to predict by the methods of temporal difference, *Mach. Learning* **3** (1988) 9–44.
[24] H. Takagi and I. Hayashi, NN-driven fuzzy reasoning, *Internat. J. Approx. Reasoning* **5** (1991) 191–212.
[25] L.-X. Wang and J.M. Mendel, Backpropagation fuzzy system as nonlinear dynamic system identifiers, *Proc. IEEE Internat. Conf. on Fuzzy Systems*, San Diego, CA (1992) 1409–1418.
[26] L.-X. Wang and J.M. Mendel, Fuzzy basis functions, universal approximation, and orthogonal least-square learning, *IEEE Trans. Neural Networks* **3** (1992) 807–814.
[27] P.J. Werbos, Neuralcontrol and fuzzy logic: connections and designs, *Internat. J. Approx. Reasoning* **6** (1992) 185–219.
[28] B. Widrow, The original adaptive neural net broom-balancer, *Proc. Internat. Symp. on Circuits and Systems* (1987) 351–357.
[29] R.R. Yager, Implementing fuzzy logic controllers using a neural network framework, *Fuzzy Sets and Systems* **48** (1992) 53–64.
[30] L.A. Zadeh, Fuzzy logic, *IEEE Comput.* (April 1988) 83–93.