Regular Issue Papers

Reinforcement Structure/Parameter Learning for Neural-Network-Based Fuzzy Logic Control Systems

Chin-Teng Lin and C. S. George Lee

***Abstract*— This paper proposes a reinforcement neural-network-based fuzzy logic control system (RNN-FLCS) for solving various reinforcement learning problems. The proposed RNN-FLCS is constructed by integrating two neural-network-based fuzzy logic controllers (NN-FLC's), each of which is a connectionist model with a feedforward multilayered network developed for the realization of a fuzzy logic controller. One NN-FLC performs as a fuzzy predictor, and the other as a fuzzy controller. Using the temporal difference prediction method, the fuzzy predictor can predict the external reinforcement signal and provide a more informative internal reinforcement signal to the fuzzy controller. The fuzzy controller performs a stochastic exploratory algorithm to adapt itself according to the internal reinforcement signal. During the learning process, both structure learning and parameter learning are performed simultaneously in the two NN-FLC's using the fuzzy similarity measure. The proposed RNN-FLCS can construct a fuzzy logic control and decision-making system automatically and dynamically through a reward/penalty signal (i.e., a "good" or "bad" signal) or through very simple fuzzy information feedback such as "high," "too high," "low," and "too low." The proposed RNN-FLCS is best applied to the learning environment, where obtaining exact training data is expensive. The proposed RNN-FLCS also preserves the advantages of the original NN-FLC, such as the ability to find proper network structure and parameters simultaneously and dynamically and to avoid the rule-matching time of the inference engine in the traditional fuzzy logic systems. Computer simulations were conducted to illustrate the performance and applicability of the proposed RNN-FLCS.**

I. Introduction

MOST of the supervised and unsupervised learning algorithms for neural networks require precise training data sets for setting the link weights and link connectivity of the neurons for various applications [1], [2]. 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]. In this paper, we are extending our previous work on neural-network-based fuzzy logic control systems (NN-FLC) [3], [4] to the reinforcement learning problem.

For the reinforcement learning problem, training data are very rough and coarse, and 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 noninstructive 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. Reinforcement learning with prediction capabilities is much more useful than the supervised learning schemes in dynamic control problems and artificial intelligence, since the success or failure signal might only be known after a long sequence of control actions. From the biological and cognitive points of view, reinforcement learning is much closer to the modern animal learning theory [5] than the supervised learning. This is also true to the learning of many high-level intelligent skills such as how to drive a car.

The development of reinforcement learning can be roughly divided into two stages. The first stage began in the 1950's, when mathematical psychologists developed computational models to explain the learning behavior of animals and human beings [6]. They viewed learning as stochastic processes and developed the so-called stochastic learning model. At almost the same time, cyberneticians and control theorists made independent efforts on the study of stochastic learning. Their work basically used deterministic automata as a model for learning systems operating in stationary random environments, and later the model was generalized to use stochastic automata [7]. More details on the stochastic learning automata can be found in [9]. At this stage, most of the learning models were "nonassociative," since there was no input to the learning system except the reinforcement signal. A typical example is the two-armed bandit problem [8]. Representative of the second stage development of reinforcement learning is the *associative reinforcement learning,* in which people tried to

Manuscript received July 16, 1992; revised February 22, 1993. This work was supported in part by the National Science Foundation under Grant CDR 8803017 to the Engineering Research Center for Intelligent Manufacturing Systems and a Grant from the Ford Foundation.

C.-T. Lin is with the Department of Control Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C.

C. S. G. Lee is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

IEEE Log Number 9213143.

associate an input pattern with output patterns according to a reinforcement signal. This was stimulated by the theory proposed by Klopf [10]. Inspired by Klopf's work and earlier simulation results [11], Barto and his colleagues used neuron-like adaptive elements to solve difficult learning control problems with only reinforcement signal feedback [12]. They also proposed the associative reward penalty (A_{R-P}) algorithm for adaptive elements called A_{R-P} elements [13], and proposed several generalizations of the A_{R-P} algorithm [14]. Williams formulated the reinforcement learning problem as a gradient-following procedure [15], and identified a class of algorithms, called REINFORCE algorithms, that possess the gradient ascent property. However, these algorithms still do not include the full A_{R-P} algorithms.

In this paper, we shall apply the technique of associative reinforcement learning to our proposed reinforcement NN-FLC learning system. The proposed learning system can construct a fuzzy logic control and decision system automatically and dynamically through a reward penalty signal (i.e., good/bad signal) or through very simple fuzzy feedback information such as "high," "too high," "low," and "too low." Moreover, there is a possibility of a long time delay between an action and the resulting reinforcement feedback information. To achieve the goal of solving reinforcement learning problems in fuzzy logic systems, a Reinforcement Neural-Network-Based Fuzzy Logic Control System (RNN-FLCS) is proposed which consists of two closely integrated NN-FLC's. One NN-FLC, the *action network,* is used for the fuzzy logic controller, it can choose a proper action or decision according to the current input vector. Its functions are the same as those proposed in [3], [4], and the major difference is that there is no "teacher" to indicate output errors for the action network to learn in the reinforcement learning problem. The other NN-FLC, the *evaluation network,* is used as the *fuzzy predictor,* and it performs the single- or multistep prediction of the scalar external reinforcement signal. The fuzzy predictor provides the action network with more informative and beforehand internal reinforcement signals for learning. Structurally, these two NN-FLC's share the first two layers of the original NN-FLC in [3]; that is, they use the same distributed representation of input patterns. This representation is the overlapping type and is dynamically adjustable through the learning process.

Associated with the proposed RNN-FLCS 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 multistep prediction. With the knowledge of output errors, the on-line supervised structure/parameter learning algorithm developed in [4] can be applied 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 on-line learning algorithm described in [4]. 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 [4]. It can determine the proper network size, connections, and parameters of an RNN-FLCS dynamically through an external reinforcement signal. Moreover, learning can proceed even in the period without any external reinforcement feedback. The RNN-FLCS also maintains the human-understandable structure of the NN-FLC, such that IF–THEN type expert knowledge can be easily incorporated into the fuzzy logic controller or the fuzzy predictor, which is basically a model of the environment or the controlled plant. 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 II, the basic structure and functions of our previously proposed NN-FLC are described [3], [4]. The structure of the proposed NN-FLCS and the corresponding reinforcement structure/parameter learning algorithm with single-step prediction capability are presented in Section III to solve simpler reinforcement learning problems. In Section IV, the multistep fuzzy predictor is proposed to perform multistep prediction in more complex reinforcement learning problems in which there is a long time delay between an action and the resultant reinforcement signal. In Section V, the cart-pole balancing problem is simulated to demonstrate the capabilities of the proposed RNN-FLCS. Finally, conclusions are summarized in Section VI.

II. Neural-Network-Based Fuzzy Logic Controller (NN-FLC)

This section introduces the structure and functions of our previously proposed Neural-Network-Based Fuzzy Logic Controller (NN-FLC) [3], [4], which is a basic component of the proposed RNN-FLCS. The learned NN-FLC functions as a connectionist neural-network-based fuzzy logic control and decision-making system. Fig. 1 shows the basic configuration of a fuzzy logic controller which is composed of three major components: fuzzifier, fuzzy rule base and inference engine, and defuzzifier. The fuzzifier performs the function of fuzzification that converts input data from an observed input space into proper linguistic values of fuzzy sets through predefined input membership functions. The rule base consists of a set of fuzzy logic rules in the form of "IF–THEN" to describe the control policy of expert knowledge. The inference engine is to match the output of the fuzzifier with the fuzzy logic rules and perform fuzzy implication and approximate reasoning to decide a fuzzy control action. Finally, the defuzzifier performs the function of defuzzification to yield a nonfuzzy (crisp) control action from an inferred fuzzy control action through predefined output membership functions. More detailed descriptions of the concepts and definitions of a fuzzy logic controller can be found in [3], [4], [16], [17]. A major problem of designing a fuzzy logic controller is determining the proper input/output membership functions and fuzzy logic rules. Based on the basic structure and concepts of the fuzzy logic controller, an NN-FLC with

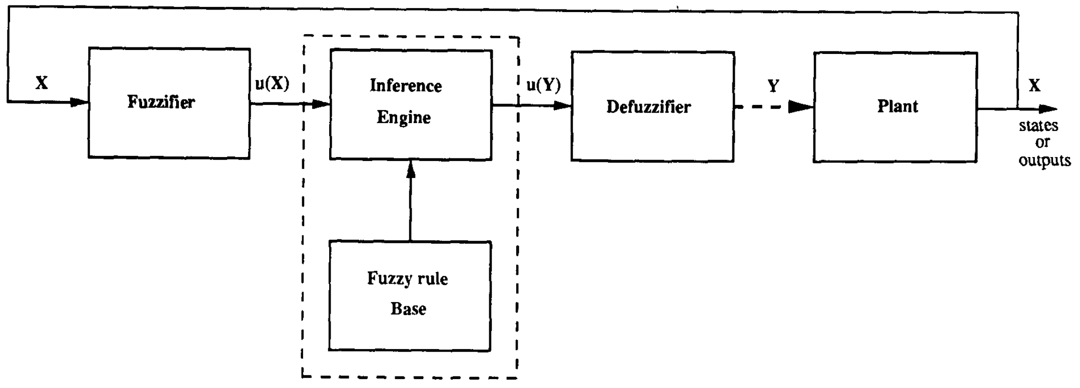


Fig. 1. General model of a fuzzy logic controller and decision making system.

a connectionist structure has been proposed [3] to realize the traditional fuzzy logic controller with learning abilities. The NN-FLC is a feedforward multilayered network that integrates the basic elements and functions of a traditional fuzzy logic controller (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. 2 shows the structure of our NN-FLC, which is described in [3]. 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 the respective linguistic variable. Actually, a layer-two node can be either a single node that performs a simple membership function (e.g., a triangular-shaped function or a bell-shaped function) or multilayered nodes (a subneural net) that perform a complex membership function (e.g., in an acoustic cue detector [4]). Hence, the total number of layers in this connectionist model could be more than five. 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. Links at layers three and four function as a *connectionist inference engine* [3], [4], which avoids the rule-matching process. Layer-three links define the preconditions of the rule nodes, and layer-four links define the consequences 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 below in Sections III and IV.

With this five-layered structure of the proposed connectionist model, a node's basic functions 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 (see Fig. 3). 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:

$$\text{net} - \text{input} = f(u_1^k, u_2^k, \cdots, u_p^k; w_1^k, w_2^k, \cdots, w_p^k) \quad (1)$$

where u_i^k represents an ith input signal from 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 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:

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

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

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

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,

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

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

•*Layer 2:* If we use a single node to perform a simple membership function, then the output function of this node should be this membership function. For example, for a bell-shaped function

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

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. Hence, the link weight at layer two (w_{ij}^2) can be interpreted as m_{ij}. If we use a set of nodes to perform a membership function, then the function of each node can be in the standard form as (3),

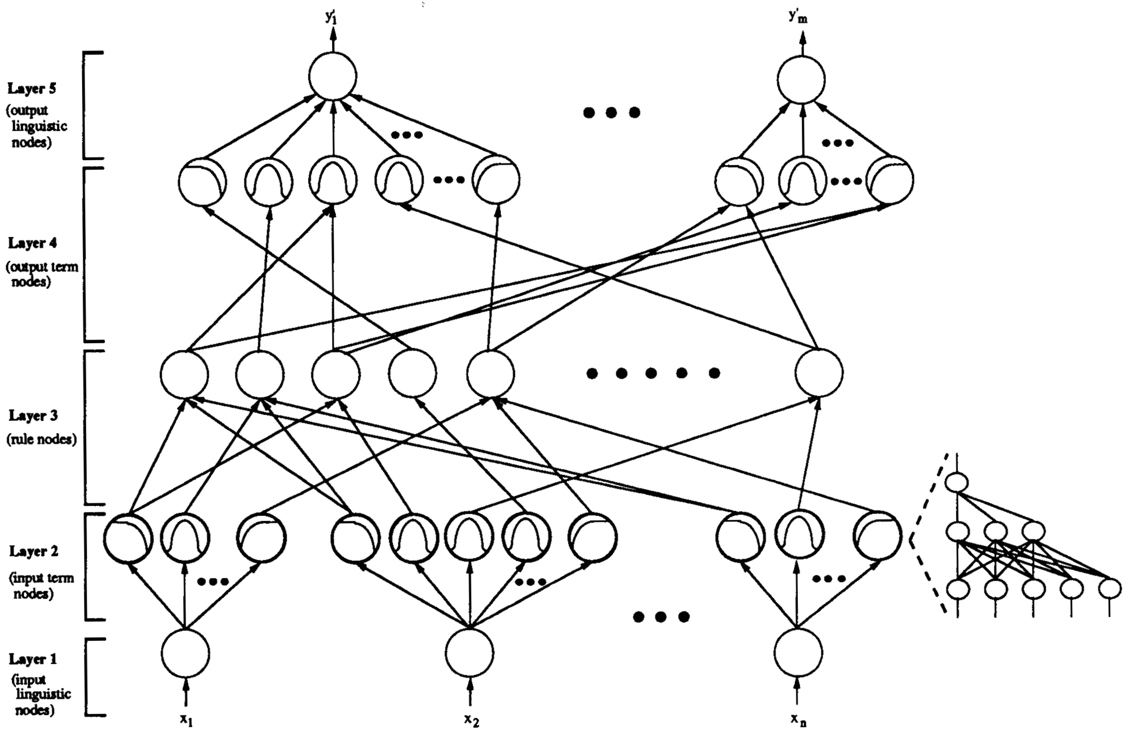


Fig. 2. Proposed neural-network-based fuzzy logic controller (NN-FLC).

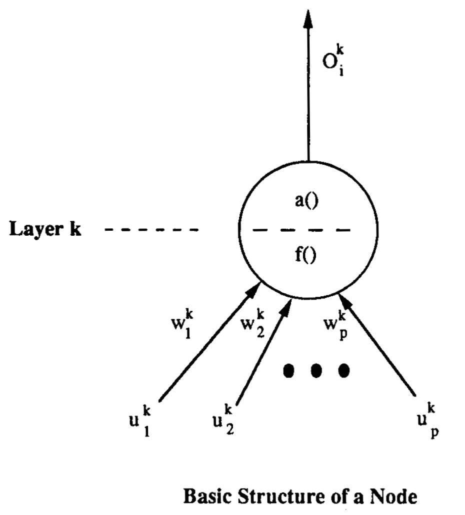


Fig. 3. Basic structure of a node in a neural network.

and the whole subnet is trained off-line to perform the desired membership function by a standard learning algorithm (e.g., backpropagation [2]).

•*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

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

The link weight in layer three (w_i^3) is then unity. Other possibilities for performing the fuzzy AND operation are "product" or "softmin" operator (a soft version of the min operator [33]). Although these operators require more computations than the min operator, they are differentiable and suitable for the derivation of a learning algorithm.

•*Layer 4:* The links at layer four should perform the fuzzy OR operation to integrate the fired rules which have the same consequent:

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

Hence, the link weight $w_i^4 = 1$.

•*Layer 5:* The nodes in this layer transmit the decision signal out of the network. These nodes and the layer-five links attached to them act as the defuzzifier. If m_{ij}^5's and σ_{ij}^5's are the centers and the widths of the membership functions, respectively, then the following functions can be used to simulate the *center of area* defuzzification method [16], [17]:

$$f = \sum w_{ij}^5 u_i^5 = \sum (m_{ij}\sigma_{ij}) u_i^5 \text{ and } a = \frac{f}{\sum \sigma_{ij} u_i^5}. \tag{8}$$

Here the link weight at layer five (w_{ij}^5) is $m_{ij}\sigma_{ij}$.

Two complementary learning schemes were proposed to set up the NN-FLC in [3] and [4]. The on-line supervised learning algorithm performs very well when the training data are available on-line [4], while the two-phase hybrid learning algorithm is superior when sets of training data are available off-line [3]. However, both learning schemes require precise training data to indicate the exact desired output, and then use the precise training data to compute the output errors for training the whole network. Unfortunately, such detailed and precise training data may be very expensive or even impossible to obtain in some real-world applications because the controlled system may only be able to provide the learning algorithm with a reinforcement signal such as a binary decision of right/wrong of the current controller/decision maker. To train a network with this kind of evaluative feedback, two NN-FLC's need to be integrated into the structure of the proposed RNN-FLCS with the corresponding reinforcement learning algorithm developed in the following sections. One NN-FLC in the proposed RNN-FLCS functions as a fuzzy controller, and the other NN-FLC as a fuzzy predictor. The reinforcement learning algorithm combines the structure learning and the parameter learning to determine optimal centers (m_{ij}'s) and widths (σ_{ij}'s) of the term nodes in layers two and four. At the same time, 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. All these learning algorithms will be performed on both NN-FLC's simultaneously and only conducted by a reinforcement signal feedback from the external environment.

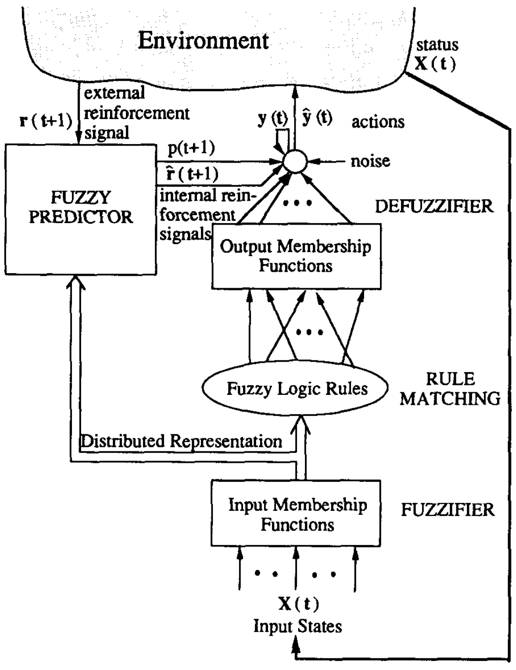


Fig. 4. Proposed reinforcement neural-network-based fuzzy logic control system (RNN-FLCS).

III. Structure/Parameter Learning Algorithm for the RNN-FLCS with a Single-Step Fuzzy Predictor

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 extensive 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 multivalued 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 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.

The precise computation of the reinforcement signal highly depends on the nature of the environment and is assumed to be unknown to the learning system. It could be a deterministic or stochastic function of the input produced by the environment and the output it receives from the network. There are three classes of reinforcement learning problems. First, for the simplest case, the reinforcement signal is always the same for a given input/output pair; hence, the network can learn a definite input/output mapping. Moreover, the reinforcement signals and input patterns do not depend on previous network output. For example, the parity learning problem and the symmetry learning problem [18] are in this class. Second, in a stochastic environment, a particular input/output pair determines only the probability of positive reinforcement. However, this probability is fixed for each input/output pair and, again, the reinforcement signal and input sequence do not refer to past history. This class includes the nonassociative reinforcement learning problem, in which there is no input, and we need to determine the best output pattern with the highest probability of positive reinforcement from only a finite set of trials. A typical example is the two-armed bandit problem [8]. Third, for the most general case, the environment is itself governed by a complicated dynamical process, and both the reinforcement signal and input patterns may depend on the past network output. For example, in a chess game, the environment is actually another player, and the network only receives a reinforcement signal (win or lose) after a long sequence of moves.

To resolve the three different classes of reinforcement learning problems, a new structure, called the reinforcement neural-network-based fuzzy logic control system (RNN-FLCS), is proposed. The proposed RNN-FLCS, as shown in Fig. 4, integrates two NN-FLC's into a learning system: one NN-

FLC for the fuzzy controller and the other NN-FLC for the fuzzy predictor. These two NN-FLC's 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. 2. 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 output as shown in Fig. 2, although only one output node is shown in Fig. 4. In the multioutput 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 multistep 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.

In this section, a reinforcement structure/parameter learning algorithm is proposed to solve the first and second classes of the reinforcement learning problem on the proposed RNN-FLCS using a single-step fuzzy predictor. Since the third class of the reinforcement learning problem is more difficult, a more powerful multistep fuzzy predictor is necessary for the RNN-FLCS. This will be discussed in Section IV.

A. Stochastic Exploration

In this section, 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,

$$\Delta m_i \propto \frac{\partial r}{\partial m_i}. \tag{9}$$

To determine $\frac{\partial r}{\partial m_i}$, we need to know $\frac{\partial r}{\partial y}$, 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 $\frac{\partial r}{\partial y}$ can only be estimated. If we can estimate $\frac{\partial r}{\partial y}$, then the on-line supervised structure/parameter learning algorithm [4] can be directly applied to 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 II. Such an approach has also been used in other reinforcement learning algorithms [12]–[15], [18]–[20] and is consistent with the closely related theory of stochastic learning automata[9].

In our learning algorithm, the gradient information, $\frac{\partial r}{\partial y}$, is also estimated by the stochastic exploratory method [19]. In particular, the intuitive idea behind the multiparameter distributions suggested by Williams [15] 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:

$$\sigma(t) = \frac{k}{2}[1 - \tanh(p(t))] = \frac{k}{1 + e^{2p(t)}} \tag{10}$$

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 $r(t)$. Equation (10) 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 (10). 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 multiparameter 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 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 multiparameter distribution approach reduces to the single-parameter distribution approach, which has been widely used in the reinforcement learning algorithms [11]–[14]. Once the variance has been decided, the actual output of the stochastic node can be set as:

$$\hat{y}(t) = N(y(t), \sigma(t)). \tag{11}$$

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

$$f(\hat{y}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\hat{y}-y)^2}{2\sigma^2}}. \tag{12}$$

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:

$$\begin{aligned}\frac{\partial r}{\partial y} &\approx [r(t) - p(t)]\left[\frac{\hat{y}(t-1) - y(t-1)}{\sigma(t-1)}\right] \\ &\equiv [r(t) - p(t)]\left[\frac{\hat{y} - y}{\sigma}\right]_{t-1}\end{aligned} \tag{13}$$

where the subscript $t-1$ represents the time displacement, and σ is a scaling factor. The time displacements in (13) and the following equations reflect the assumption that the reinforcement signal (which may be the "predicted" reinforcement signal in the multistep fuzzy predictor) at time step t depends on the input and actions chosen at time step $t-1$. In (13), the term $\frac{\hat{y}-y}{\sigma}$ is the normalized difference between the actual and 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 $y(t-1)$. Equation (13) 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 (13) is frequently adopted in the stochastic exploration techniques [19].

After the gradient information is available, we have transformed the reinforcement learning problem to the supervised learning problem and can apply the on-line supervised structure/parameter learning algorithm in [4] to develop the following reinforcement structure/parameter learning algorithm for the action network in the proposed RNN-FLCS. A detailed derivation and description of the on-line supervised structure/parameter learning algorithm can be found in [4].

One important characteristic of the on-line supervised structure/parameter learning algorithm is that it can learn both the network structure and parameters simultaneously. Learning the network structure includes deciding the proper number of output term nodes in Layer 4 and the proper connections between the nodes in Layers 3 and 4 of an NN-FLC. This learning also decides the coarse of the output fuzzy partitions and finds the correct fuzzy logic rules. Learning the network parameters includes adjusting the node parameters in Layers 2 and 4. This corresponds to learning input and output membership functions. The flowchart of this on-line learning algorithm is shown in Fig. 5. Given the gradient error information in (13), the proposed learning algorithm first decides whether or not to perform the structure learning based on the previously proposed fuzzy similarity measures [4] of the output membership functions. If structure learning is necessary, then the proposed learning algorithm will further decide whether to add a new output term node (a new membership function); it will also change the consequenct of some fuzzy logic rules properly. After the structure learning process, the parameter learning will be performed to adjust the current membership functions. This structure/parameter learning will be repeated for each real-time incoming internal reinforcement signal, which appears either with the same frequency as the external reinforcement signal (in the single-step prediction problem) or with much higher frequency than the external reinforcement signal (in the multistep prediction problem). When the structure/parameter training loop is complete, rule combination [3] is performed to find the minimum node representation of fuzzy logic rules. This is the final step in the process.

Before entering the structure/parameter learning loop, as shown in Fig. 5, we need to perform two kinds of initialization: structure initialization and parameter initialization. In the structure initialization, the desired coarse of input fuzzy partitions (i.e., the size of the term set of each input linguistic variable) and the initial guess of output fuzzy partitions must be provided from the outside world. Before this network is trained, an initial form of the network is constructed and, during the learning process, new nodes may be added and some connections changed. Finally, after the learning process, some nodes and links of the network will be deleted or combined to form the final structure of the network. In its initial form (see Fig. 2), there are $\prod_i |T(x_i)|$ rule nodes with the input of each rule node coming from one possible combination of the terms of input linguistic variables with the constraint that only one term in a term set can be a rule node's input. Here, $|T(x_i)|$ indicates the number of terms of x_i (i.e., the number of fuzzy partitions of input state linguistic variable x_i). Thus, 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. Furthermore, there is only one link between a rule node and an output linguistic variable. This link is connected to a term node of the output linguistic variable. The initial candidate (term node) of the consequent of a rule node can be assigned by an expert (if possible) or chosen randomly. A suitable term in each output linguistic variable's term set will be chosen for each rule node after the learning process. With the initial network structure, the parameters in this structure should be initialized. The parameter initialization decides the initial membership functions of input/output linguistic variables. Theoretically, they can be set randomly; however, a more efficient way is to use identical membership functions such that their domains can cover the region of corresponding input/output space *evenly* according to the given initial coarse of fuzzy partitions. This initilization process is used for both the action network and the evaluation network, which can be a single-step or multistep fuzzy predictor.

After the initialization process, the learning algorithm enters the training loop in which each loop corresponds to an incoming internal reinforcement signal. Basically, the idea of

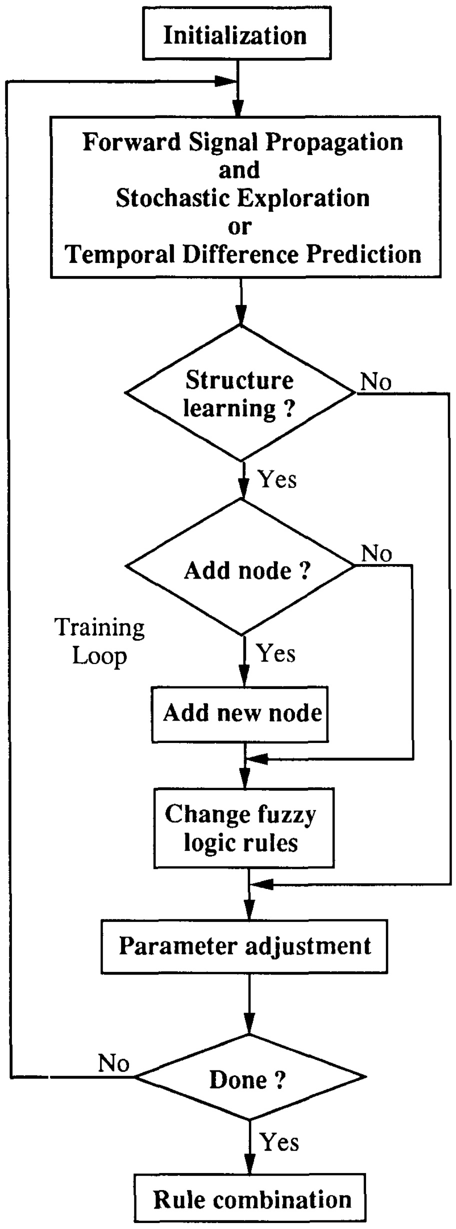


Fig. 5. Flowchart of the proposed reinforcement structure/parameter learning algorithm.

backpropagation [2] is used here to find the errors of node output in every layer except the output layer. These errors are then analyzed by the fuzzy similarity measure to perform structure and/or parameter adjustments. The detailed learning rules are derived.

The goal 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 $\frac{\partial r}{\partial y}$. Then, starting at the output nodes, a backward pass computes $\frac{\partial r}{\partial f}$ 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:

$$\Delta w \propto \frac{\partial r}{\partial w} \tag{14}$$

$$w(t+1) = w(t) + \eta\left(\frac{\partial r}{\partial w}\right) \tag{15}$$

where η is the learning rate, and

$$\begin{aligned}\frac{\partial r}{\partial w} &= \frac{\partial r}{\partial(\text{net}-\text{input})}\frac{\partial(\text{net}-\text{input})}{\partial w} = \frac{\partial r}{\partial f}\frac{\partial f}{\partial w}\\ &= \frac{\partial r}{\partial a}\frac{\partial a}{\partial f}\frac{\partial f}{\partial w}.\end{aligned} \tag{16}$$

To show the learning rule, we shall show the computations of $\frac{\partial r}{\partial f}$, 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 (16), (13), and (8), the adaptive rule of the center m_i is derived:

$$\begin{aligned}\frac{\partial r}{\partial m_i} &= \frac{\partial r}{\partial a}\frac{\partial a}{\partial f^5}\frac{\partial f^5}{\partial m_i} = [r(t)-p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}\\ &\cdot \left[\frac{\sigma_i u_i}{\sum \sigma_i u_i}\right]_{t-1}\end{aligned} \tag{17}$$

Hence, the expected updated amount of the center is:

$$\Delta m_i(t) = \eta[r(t)-p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}\left[\frac{\sigma_i u_i}{\sum \sigma_i u_i}\right]_{t-1}. \tag{18}$$

Similarly, using (6), (13), and (8), the adaptive rule of the width σ_i is derived:

$$\begin{aligned}\frac{\partial r}{\partial \sigma_i} &= \frac{\partial r}{\partial a}\frac{\partial a}{\partial f^5}\frac{\partial f^5}{\partial \sigma_i} = [r(t)-p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}\\ &\cdot \left[\frac{m_i u_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i) u_i}{(\sum \sigma_i u_i)^2}\right]_{t-1}.\end{aligned} \tag{19}$$

Hence, the expected updated amount of the width parameter is:

$$\begin{aligned}\Delta \sigma_i(t) &= \eta[r(t)-p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}\\ &\cdot \left[\frac{m_i u_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i) u_i}{(\sum \sigma_i u_i)^2}\right]_{t-1}.\end{aligned} \tag{20}$$

The error to be propagated to the preceding layer is:

$$\delta^5(t) = \frac{\partial r}{\partial f^5} = \frac{\partial r}{\partial a}\frac{\partial a}{\partial f^5} = [r(t)-p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}. \tag{21}$$

Fuzzy Similarity Measure: In this step, the system decides whether the current structure should be changed according to the expected updated amount of the center and width parameters [(18) and (20)]. To do this, the expected center and width are, respectively, computed:

$$m_{i\text{-new}} = m_i(t) + \Delta m_i(t) \qquad (22)$$
$$\sigma_{i\text{-new}} = \sigma_i(t) + \Delta\sigma_i(t).$$

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 [4] determines the similarity between two fuzzy sets. If A and B are two fuzzy sets with bell-shaped membership functions, then:

$$\mu_A(x) = e^{-\frac{(x-m_1)^2}{\sigma_1^2}} \quad \text{and} \quad \mu_B(x) = e^{-\frac{(x-m_2)^2}{\sigma_2^2}}. \qquad (23)$$

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

$$E(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{\sigma_1\sqrt{\pi} + \sigma_2\sqrt{\pi} - M(A \cap B)} \qquad (24)$$

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

$$\begin{aligned} |A \cap B| = &\frac{1}{2}\frac{h^2(m_2 - m_1 + \sqrt{\pi}(\sigma_1 + \sigma_2))}{\sqrt{\pi}(\sigma_1 + \sigma_2)} \\ &+ \frac{1}{2}\frac{h^2(m_2 - m_1 + \sqrt{\pi}(\sigma_1 - \sigma_2))}{\sqrt{\pi}(\sigma_2 - \sigma_1)} \\ &+ \frac{1}{2}\frac{h^2(m_2 - m_1 - \sqrt{\pi}(\sigma_1 - \sigma_2))}{\sqrt{\pi}(\sigma_1 - \sigma_2)} \end{aligned} \qquad (25)$$

where $h(x) = \max\{0, x\}$.

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

$$\begin{aligned} \text{degree}(i,t) &= E[M(m_{i\text{-new}}, \sigma_{i\text{-new}}), \\ &\qquad M(m_{i\text{-closest}}, \sigma_{i\text{-closest}})] \\ &= \max_{1 \leq j \leq k} E[M(m_{i\text{-new}}, \sigma_{i\text{-new}}), M(m_j, \sigma_j)] \end{aligned} \qquad (26)$$

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 criterion such that the *lower similarity* is allowed in the initial stages of the learning. According to this judgment, degree(i,t) is first compared to the similarity criterion. If there is not enough similarity, then a new term node (new membership function) with the expected parameters is built because, in this case, all the current membership functions are too different from the expected one. A new node with the expected membership function is necessary, and the output connections of some just firing rule nodes should be changed to point to this new term node through the structure learning process. If no new term node is necessary, the learning algorithm will then check if the ith term node is the i-closest node. If this is false, some of the just fired 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; only the parameter learning should be performed by the standard backpropagation algorithm. The structure learning process is given later.

Structure Learning: 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 consequences should be changed, we set a *firing strength threshold* β. Only the rules whose firing strengths are higher than this threshold are treated as *real firing* rules. Only the real firing rules are considered for changing their consequent, since only these rules are fired strongly enough to contribute to the results of judgment. Assuming that the term node $M(m_i, \sigma_i)$ in layer 4 receives input from rule nodes $1 \cdots l$ in layer 3, whose corresponding firing strengths are a_i^3's,$i = 1 \cdots l$, then:

IF $a_i^3(t) \geq \beta$, THEN change the consequent of the ith rule node

from $M(m_i, \sigma_i)$ to $M(m_{i\text{-new}}, \sigma_{i\text{-new}})$.

To utilize the error signal more efficiently, the following *fine tuning* can be performed. Let

$$k_1 = \min\left(1, \sum_{a_i^3 \geq \beta} a_i^3\right) \quad \text{and} \quad k_2 = \min\left(1,, \sum_{a_i^3 < \beta} a_i^3\right). \qquad (27)$$

Then,

$$(\Delta m_i)_{\text{extra}} = \eta' k_2 \Delta m_i = \eta' k_2 \eta [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma}\right]_{t-1} \cdot \left[\frac{\sigma_i u_i}{\sum \sigma_i u_i}\right]_{t-1}$$

$$(\Delta\sigma_i)_{\text{extra}} = \eta' k_2 \Delta\sigma_i = \eta' k_2 \eta [r(t) - p(t)] \left[\frac{\hat{y} - y}{\sigma}\right]_{t-1}$$

$$\cdot\left[\frac{m_i u_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i) u_i}{(\sum \sigma_i u_i)^2}\right]_{t-1} \tag{28}$$

$$(\Delta m_{i-\text{closest}})_{\text{extra}} = \eta' k_1 (m_{i-\text{new}} - m_{i-\text{closest}})$$
$$(\Delta \sigma_{i-\text{closest}})_{\text{extra}} = \eta' k_1 (\sigma_{i-\text{new}} - \sigma_{i-\text{closest}})$$

where $0 \le \eta' < \eta < 1$. The subscript "extra" denotes the updated amount in addition to those calculated from other possible error signals. The cutoff value β is chosen empirically. In most cases, a consequent label, which is found to cause a large output error, is usually supported by one or a few strongly fired rules. Hence, it is not too difficult to find a proper β value. A β value in the range [0.5, 0.8] usually leads to good learning results. However, it is possible that several weakly fired rules, which share one consequent label, might affect the output substantially. In this extreme case, the structure learning rule might not be able to change the consequent of these rules properly. This case rarely happens because it means that a set of rules sharing one consequent is assigned to a wrong consequent label simultaneously in the learning process. Three possible methods can be used to solve this problem. First, the β value can be changed adaptively such that it will be lowered when all the rules to a "wrong" consequent label are fired weakly. Second, a cutoff is put on the effect of the Layer 4 consequent label rather than using such a cutoff at the Layer 3 output. Third, the cutoff β could be eliminated by weighting the changes caused in a Layer 4 node or by the sum of degrees of rules feeding into it with suitable normalization. This weighting scheme will automatically cause large changes in the important rules only and will do so in a smooth and graceful manner. These suggested modifications on the structure learning algorithm need further studies.

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

$$\delta_i^4 = \frac{\partial r}{\partial f_i} = \frac{\partial r}{\partial a_i}\frac{\partial a_i}{\partial f_i} = \frac{\partial r}{\partial(\text{net} - \text{input})^5} \cdot \frac{\partial(\text{net} - \text{input})^5}{\partial a_i} \tag{29}$$

where, from (8),

$$\frac{\partial(\text{net} - \text{input})^5}{\partial a_i} = \frac{\partial f^5}{\partial u_i^5} = \frac{m_i\sigma_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i)\sigma_i}{(\sum \sigma_i u_i)^2} \tag{30}$$

and, from (21),

$$\frac{\partial r}{\partial(\text{net} - \text{input})^5} = \frac{\partial r}{\partial f^5} = \delta^5 = [r(t) - p(t)] \cdot \left[\frac{\hat{y} - y}{\sigma}\right]_{t-1}. \tag{31}$$

Hence, the error signal is:

$$\delta_i^4(t) = [r(t) - p(t)]\left[\frac{\hat{y} - y}{\sigma}\right]_{t-1} \cdot \left[\frac{m_i\sigma_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i)\sigma_i}{(\sum \sigma_i u_i)^2}\right]_{t-1}. \tag{32}$$

In the multioutput case, the computations in layers five and four are exactly the same as the ones 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 (7), this error signal can be derived:

$$\delta_i^3(t) = \frac{\partial r}{\partial f_i} = \frac{\partial r}{\partial a_i}\frac{\partial a_i}{\partial f_i} = \frac{\partial r}{\partial(\text{net} - \text{input})^4}\frac{\partial(\text{net} - \text{input})^4}{\partial a_i} = \delta_i^4(t)\frac{\partial f^4}{\partial u_i^4} = \delta_i^4(t). \tag{33}$$

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 consequences of a rule node; that is, the error of a rule node is the summation of the errors of its consequent.

•*Layer 2:* Using (16) and (5), the adaptive rule of m_{ij} is derived:

$$\frac{\partial r}{\partial m_{ij}} = \frac{\partial r}{\partial a_i}\frac{\partial a_i}{\partial f_i}\frac{\partial f_i}{\partial m_{ij}} = \frac{\partial r}{\partial a_i} e^{f_i}\frac{2(u_i - m_{ij})}{\sigma_{ij}^2} \tag{34}$$

where, from (33),

$$\frac{\partial r}{\partial a_i} = \sum_k \frac{\partial r}{\partial(\text{net} - \text{input})^k}\frac{\partial(\text{net} - \text{input})^k}{\partial a_i} \tag{35}$$

$$\frac{\partial r}{\partial(\text{net} - \text{input})^k} = \frac{\partial r}{\partial f_k^3} = \delta_k^3 \tag{36}$$

and, from (6),

$$\frac{\partial(\text{net} - \text{input})^k}{\partial a_i} = \frac{\partial f^3}{\partial u_i^3} = \begin{cases} 1 & \text{if } u_i^3 = \min \text{ (inputs of rule node } k); \\ 0 & \text{otherwise.} \end{cases} \tag{37}$$

Hence,

$$\frac{\partial r}{\partial a_i} = \sum_k q_k(t) \tag{38}$$

where the summation is performed over the rule nodes that a_i feeds into, and

$$q_k(t) = \begin{cases} \delta_k^3(t) & \text{if } a_i \text{ is minimum in } k\text{th rule node's input;} \\ 0 & \text{otherwise.} \end{cases} \tag{39}$$

So, the adaptive rule of m_{ij}'s is:

$$m_{ij}(t+1) = m_{ij}(t) - \eta\left[\frac{\partial r}{\partial a_i}\right]_t \left[e^{f_i}\frac{2(u_i - m_{ij})}{\sigma_{ij}^2}\right]_{t-1}. \tag{40}$$

Similarly, using (16), (5), and (35)–(39), the adaptive rule of σ_{ij} is derived:

$$\frac{\partial r}{\partial \sigma_{ij}} = \frac{\partial r}{\partial a_i}\frac{\partial a_i}{\partial f_i}\frac{\partial f_i}{\partial \sigma_{ij}} = \left[\frac{\partial r}{\partial a_i}\right]_t \left[e^{f_i}\frac{2(u_i - m_{ij})^2}{\sigma_{ij}^3}\right]_{t-1}. \tag{41}$$

Hence, the adaptive rule of σ_{ij} becomes:

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) - \eta \left[\frac{\partial r}{\partial a_i}\right]_t \left[e^{f_i} \frac{2(u_i - m_{ij})^2}{\sigma_{ij}^3}\right]_{t-1}. \quad (42)$$

Since the min operator used in layer 3 is nondifferentiable, the learning rule [(37) and (39)] will cause a Dirac delta type of function. This can be avoided if the softmin operator [33] is adopted. In this case, a form of leaky learning will take place at the antecedent level of the network and may lead to a faster learning with a less number of iterations. However, due to more complex computations of the softmin operator, the actual learning time in each iteration will be longer. Hence, for practical considerations, the min operator is selected for performing the fuzzy AND operation in layer 3.

The whole learning procedure is summarized by the flowchart in Fig. 5. The proposed reinforcement learning algorithm provides a novel on-line scheme to combine the structure learning and the parameter learning such that they can be performed simultaneously. Finally, it should be noted that this backpropagation algorithm can be easily extended to train the membership function implemented by a subneural net instead of a single-term node in layer two since, from the analysis, the error signal can be propagated to the output node of the subneural net. Then, using a similar backpropagation rule in this subneural net, the parameters in this subneural net can be adjusted.

B. Single-Step Fuzzy Predictor

We shall use an NN-FLC to develop a single-step 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. 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 in the first and second classes of the reinforcement learning problem. If the fuzzy predictor can produce a signal $p(t)$, which is the prediction of $r(t)$ but 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 multiparameter probability distribution (10). 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 (13) by the action network. The single-step prediction is the extreme case of the multistep prediction which will be presented in the next section. 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 in [4] for the NN-FLC with a single output node. The goal to train the single-step fuzzy predictor is to minimize the squared error prediction:

$$E = \frac{1}{2}(r(t) - p(t))^2 \quad (43)$$

where $r(t)$ represents the desired output (real external reinforcement signal), and $p(t)$ is the current output (predicted reinforcement signal). Then, the gradient information can be easily derived:

$$\frac{\partial E}{\partial p} = p(t) - r(t). \quad (44)$$

Similar to the learning rule developed in the last subsection, we can derive the structure/parameter learning algorithm for the single-step fuzzy predictor using the following general parameter learning rule:

$$w(t+1) = w(t) + \eta\left(-\frac{\partial E}{\partial w}\right) \quad (45)$$

where w is the adjustable parameters in the fuzzy predictor. The learning equations are the same as (16)–(42) if $\frac{\partial r}{\partial y}$ is replaced by $\left(-\frac{\partial E}{\partial p}\right)$ and the effects caused by this replacement are properly updated; that is, all the terms $[r(t) - p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}$ in (16)–(42) are replaced with the term $[r(t) - p(t)]$.

In the RNN-FLCS, the action and evaluation networks are to be trained together. In principle, we could perform the learning for both networks at the same time since they can be treated individually. Although they share the same fuzzifier, they can find a common input internal distributed representation (i.e., the same input membership functions) suitable for them. However, since the action network relies on the accurate prediction of the evaluation network, it seems practical to train the fuzzy predictor first, at least partially, or to let the fuzzy predictor have a higher learning rate than the action network.

After the consequents of rule nodes are determined for both the action and evaluation networks (i.e., when the structure learning process is done and the structure will not be changed any further, but the parameter refinement may still be performed), the rule combination in [3], [4] is performed to find the minimum node representation of current fuzzy logic rules. The criteria for a set of rule nodes to be combined into a single rule node are: 1) that 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.

IV. Multistep Fuzzy Predictor

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 RNN-FLCS is to design a multistep 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 multistep 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, which are often closely related to the dynamic programming techniques [22], is used [12], [21]. 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 $x_1, x_2, \cdots, x_m, z$, 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 $p_1, p_2, \cdots, p_m$, 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:

$$\Delta w_t = \eta(p_t - p_{t-1}) \sum_{k=1}^{t-1} \lambda^{t-k-1} \nabla_w p_k \tag{46}$$

where $p_{m+1} \equiv z, 0 \leq \lambda \leq 1$, 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, $p_1, p_2, \cdots, p_{t-1}$, are altered properly according to the current temporal difference, $p_t - p_{t-1}$, to an "equal" extent. In this case, (46) 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 [23]. 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 [21].

•*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 $z_{t-1} - p_{t-1} = \sum_{k=t}^{m+1} r_k - p_{t-1} = \sum_{k=t}^{m+1}(r_k + p_k - p_{k-1})$, where p_{m+1} is defined as 0. Thus, the learning rule is:

$$\Delta w_t = \eta(r_t + p_t - p_{t-1}) \sum_{k=1}^{t-1} \lambda^{t-k-1} \nabla_w p_k. \tag{47}$$

•*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 $\gamma, 0 \leq \gamma < 1$, 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:

$$\Delta w_t = \eta(r_t + \gamma p_t - p_{t-1}) \sum_{k=1}^{t-1} \lambda^{t-k-1} \nabla_w p_k. \tag{48}$$

In applying the temporal difference procedures to the proposed RNN-FLCS, we let $\lambda = 0$ due to its efficiency and accuracy [21]. A general learning rule used for the three cases is:

$$\Delta w_t = \eta(r_t + \gamma p_t - p_{t-1}) \nabla_w p_{t-1} \tag{49}$$

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

We shall next derive the learning rule of the multistep fuzzy predictor according to (49). 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 multistep prediction problem stated. According to (49), let:

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1), \qquad 0 \leq \gamma < 1. \tag{50}$$

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

$$\Delta w(t) = \eta \hat{r}(t) \left[\frac{\partial p}{\partial w}\right]_{t-1} \tag{51}$$

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 (16)–(42). The only exception is that the error signal is different. Thus, the learning equations for the multistep fuzzy predictor are the same as in (16)–(42) 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 (50). Also, the multistep 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. 4).

The learning algorithm for the action network is the same as that derived in section III-A. However, due to the different nature of the internal reinforcement signal $\hat{r}(t)$, the learning algorithm of the action network with the multistep 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 $\frac{\partial r}{\partial y}$, as we did previously. 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 (10). Then, the actual output $\hat{y}(t)$ can be determined according to (11). Since $\hat{r}(t)$ is the prediction error, the gradient information is estimated as:

$$\frac{\partial r}{\partial y} = \hat{r}(t)\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}. \tag{52}$$

In (50), 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

$$\frac{\partial r}{\partial y} = [r(t) - [p(t-1) - \gamma p(t)]]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1} \tag{53}$$

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 $\frac{\partial r}{\partial y}$ [(53)], 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 (14)–(42), except that $[r(t)-p(t)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}$ has been replaced by the new error term $[r(t)+\gamma p(t)-p(t-1)]\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}$.

Until now, we have developed the reinforcement learning algorithm for the action network with the multistep fuzzy predictor in the multistep prediction problem. The issues of learning rate and learning order for both the action and evaluation networks and the final step to find the minimum node representation of fuzzy logic rules using the rule combination technique are the same as discussed in Section III.

One interesting advantage of using the NN-FLC as a predictor is attributed to the high-level, human-understandable meaning of the NN-FLC [3],[4]. When the RNN-FLCS works in a learning environment, the fuzzy predictor attempts to model the status-reaction relation of the environment, and this relation can be interpreted as the "IF-THEN" rules on the input/output linguistic variables. For example, the interpretation may resemble "IF the load is *a little light* and the power is *very high,* THEN the machine is *easily* over-running." Using NN-FLC as a predictor is more interesting when there are human factors in the learning environment. For example, if we use the RNN-FLCS to design an air conditioner, then the fuzzy predictor can model the user's sensitivity of feeling like this: "IF the temperature *is warm* and the humidity is *high,* THEN I feel *a little* uncomfortable." Moreover, the learned fuzzy predictor can tell us exactly what the user means by "warm," "high humidity," and "a little uncomfortable" via the learned input/output membership function, though such feelings may vary from person to person.

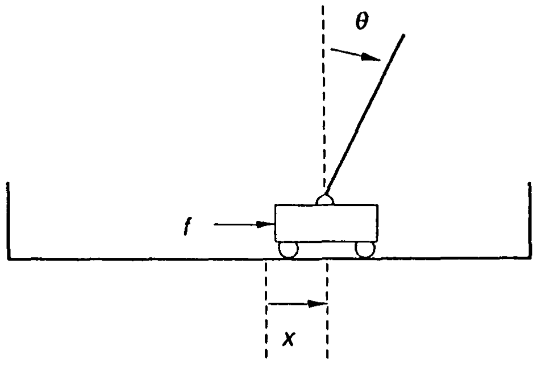


Fig. 6. The cart-pole balancing system.

A classic application of the reinforcement learning is in game theory where the "environment" is another player or players. For example, considering the RNN-FLCS for a chess game, the fuzzy predictor can learn its opponent's skill by explaining its opponent's thinking rules. The human-understandable meaning of the NN-FLC can also easily make the RNN-FLCS incorporate existent or obvious expert knowledge. This not only benefits the design of a fuzzy logic controller, but is also a great help to the fuzzy predictor in some applications. Although the membership functions are always difficult to find, the fuzzy logic rules may be obvious in some cases. Considering the air conditioner example, the rules of users' reactions to a room's status are much alike, but the standards of the feeling for high/low temperature may be quite different. In such a case, we can set up the structure of the fuzzy predictor manually according to the known fuzzy logic rules and let the input/output membership functions learn to fit different users.

V. An Illustrative Example

The proposed RNN-FLCS with multistep fuzzy predictor has been simulated on a Sun SPARC station 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 and dynamic systems to demonstrate both modern and classic control techniques [24], [25], as well as the learning control techniques of neural networks using supervised learning methods [26] or reinforcement learning methods [12], [27], [28].

As shown in Fig. 6, 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 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, the angle of the pole from an upright position (in degrees); $\dot{\theta}$, the angular velocity of the pole (in degrees/second); x, the horizontal position of the cart's center (in meters); and $\dot{x}$, the velocity 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 degrees is used here) or the cart runs into the bounds of its track (the distance is $2.4m$ from the center to both bounds of the track). The goal of this control problem is to train the RNN-FLCS 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 corresponding parameters of the cart-pole balancing system for our computer simulation are adopted from [29] with additional consideration of friction effects. This model and its parameters are also used by Barto, Sutton, and Anderson [12], [28]. The equations of motion that we used are shown below, where

1) $g = -9.8m/s^2$, acceleration due to the gravity
2) $m = 1.1$ kg, combined mass of the pole and the cart
3) $m_p = 0.1$ kg, mass of the pole
4) $l = 0.5$ m, half-pole length
5) $\mu_c = 0.0005$, coefficient of friction of the cart on the track
6) $\mu_p = 0.000002$, coefficient of friction of the pole on the cart
7) $\Delta = 0.02$, sampling interval

The constraints on the variables are $-12^\circ \le \theta \le 12^\circ, -2.4m \le x \le 2.4m$, and $-10N \le f \le 10N$. 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 $|\theta| > 12^\circ$ or $|x| > 2.4m$. 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, this cart-pole balancing problem belongs to the third class of the reinforcement learning problems discussed in Section III. Thus, a multistep fuzzy predictor is required for the RNN-FLCS. 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 RNN-FLCS 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, this cart-pole balancing problem is further categorized as the third case of the multistep prediction problem discussed in Section IV, and (49) must be used for the temporal difference prediction method. The reinforcement signal is defined as:

$$r(t) = \begin{cases} -1 & \text{if } |\theta(t)| > 12^\circ \text{ or } |x(t)| > 2.4m; \\ 0 & \text{otherwise} \end{cases} \tag{55}$$

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 ten runs by trying to use the same learning parameter values in [12]. Each run consisted of a sequence of trials; each trial

TABLE I
INPUT/OUTPUT MEMBERSHIP FUNCTIONS BEFORE AND AFTER LEARNING

Membership Functions Before and After Learning					
Linguistic Variable	Term	Parameters Before Learning		Parameters After Learning	
		Center	Width	Center	Width
x	0	-2.4	2.6	-1.98	3.11
	1	0.0	1.5	0.13	1.28
	2	2.4	2.6	2.17	4.13
$\dot{x}$	0	-20.0	20.0	-15.42	17.21
	1	0.0	10.0	-0.21	7.39
	2	20.0	20.0	17.25	16.01
θ	0	-12.0	6.0	-10.22	7.39
	1	-6.0	0.0	-5.01	4.81
	2	-3.0	3.0	-1.82	2.12
	3	0.0	3.0	0.21	1.92
	4	3.0	3.0	2.19	1.89
	5	6.0	6.0	5.31	3.87
	6	12.0	6.0	8.82	8.01
$\dot{\theta}$	0	-100.0	100.0	-84.29	84.62
	1	0.0	50.0	-0.32	39.42
	2	100.0	100.0	73.18	110.39
f	0	-10.0	6.0	-9.48	4.21
	1	-6.6	5.0	-7.89	2.89
	2	-3.3	5.0	-6.42	2.72
	3	0.0	5.0	-4.03	3.02
	4	3.3	5.0	-2.01	1.36
	5	6.6	5.0	-1.32	0.98
	6	10.0	6.0	-0.98	0.72
	7	–	–	-0.19	0.57
	8	–	–	1.12	0.33
	9	–	–	1.86	0.68
	10	–	–	2.77	1.03
	11	–	–	4.59	2.92
	12	–	–	7.21	3.57
	13	–	–	8.88	3.24
	14	–	–	10.26	4.56

$$\theta(t+1) = \theta(t) + \Delta\dot{\theta}(t) \tag{54}$$

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

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

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

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 $|x(t)| > 2.4m$. 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 (5) with initial centers and widths shown in Table I. 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) is available to this control problem. If expert knowledge is available in advance, then the initial rules can be set up correctly. In this case, we can even skip the structure learning portion in our learning algorithm, and this will greatly shorten the learning time. A study on this issue has been reported in [4]. Runs consisted of at most 50 trials, unless the duration of each run exceeded 500 000 time steps. The results of two "trials" are shown in Fig. 7. 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. The two trials shown in Fig. 7 are two failure trials. The status of system parameters in the cart-pole balancing problem is also shown in this figure. The first example in Fig. 7 is a trial in the early stage of a "run," so it failed in only about 450 time steps. From the figure showing the deviation angle, we can see that this trial failed because the pole fell outside the desired range (the pole's deviation angle is greater than 12°). At this stage, the FLC had learned to some extent in the area around the zero state of the input space since the FLC did try to decrease the pole's deviation angle before the failure occurred (see the figure showing the force). However, it still needs extensive learning in the other portion of the input space. The second example in Fig. 7 is a trial in the later stage of a "run," so it keeps the pole in the desired position longer. The system failed at around the 2800th time steps because the cart bumped into the right wall. At this stage, the FLC had learned quite well in some areas close to the zero state of the input space and it could control the pole in the correct position well. However, more learning is still necessary for the FLC to control the position of the cart. It seems that more training data from some "desired" regions of the input space (especially training data corresponding to the far ends of the track) were necessary for further training of the FLC.

In our computer simulations, a total of ten runs were performed. Among these ten runs, five of them started with zero initial conditions and the others started with randomized initial conditions. The simulation results (see Fig. 8) showed that the RNN-FLCS can learn to balance the pole within 20 trials. In most of the ten runs, the learning was completed before ten trials. It was observed that the runs starting with

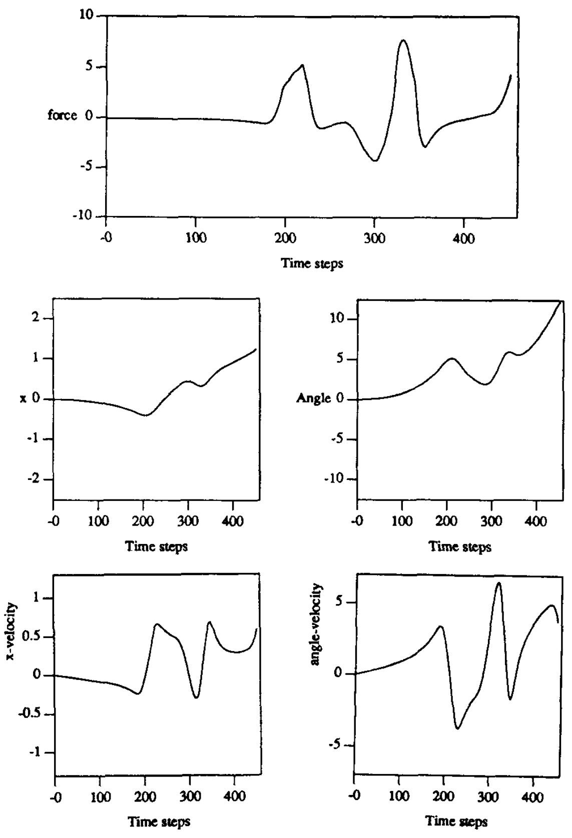


Fig. 7. Two trials in the learning process of the NN-FLCS on the cart-pole balancing problem.

a randomized initial condition usually took more trials. Fig. 9 shows the angle deviation of the pole about the center point when the cart-pole system was controlled by a well-trained RNN-FLCS. This performance is better than the results presented in [12], [28] and compatible to those in [20]. 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 [20] for each run; that is, one output membership function for each (overlapping) grid of input space. One learned fuzzy logic controller (action network) in a run is shown in Fig. 10. It has a total of 35 fuzzy logic rules. Each node in the third layer corresponds to one rule; the links to and from a rule node specify the preconditions and conclusion of the corresponding rule. The parameters of the learned input membership functions in the fuzzifier (the second layer of the action network), and the learned output membership functions in the defuzzifier (the fourth layer of the action network) are tabulated in Table I.

Similar to the simulations in [20], the adaptation capability of the proposed RNN-FLCS was tested. In a series of tests, the proposed RNN-FLCS was required to adapt to changes in the length and mass of the pole. Six tests were performed. The first two tests were to increase the original mass of the pole by a factor of 10 and 20, respectively. In the next two tests, the original pole was shortened, respectively, to 2/3 and 1/3 of the

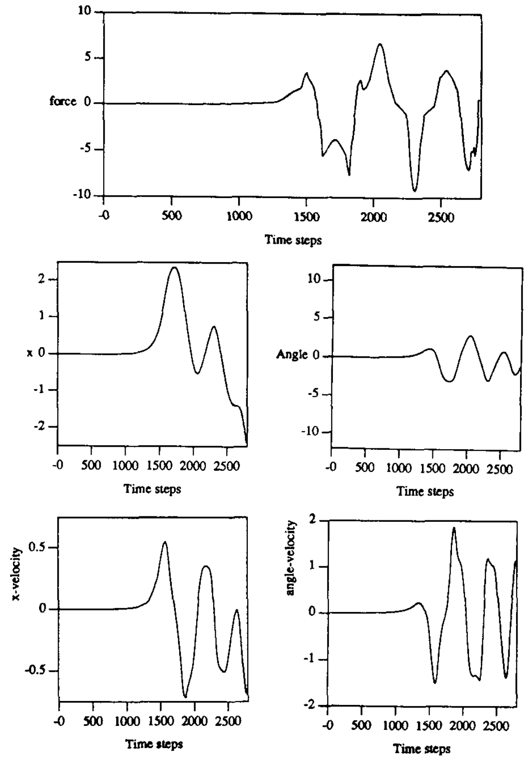


Fig. 7. Continued.

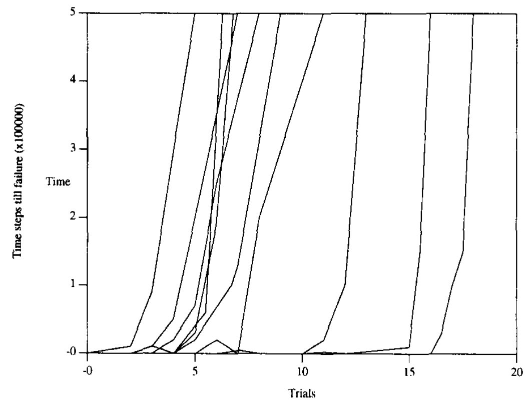


Fig. 8. Performance of the RNN-FLCS on the cart-pole balancing problem.

original pole length. The last two tests were to simultaneously reduce the length and weight of the pole to 2/3 and 1/3 of its original values, respectively. We found that the RNN-FLCS with pretrained knowledge was able to complete the control task of balancing the pole without further training/learning.

In addition to keeping the capability that learning can perform at each time step within a trial without waiting to know the actual outcome by using the multistep fuzzy predictor, the proposed RNN-FLCS has some important features over the previous work. First, distributed representation is used to represent the input vectors in the RNN-FLCS. This is achieved by the fuzzification process through the adaptive input membership functions. With the adaptive input membership functions, the input space can be considered to be divided into overlapping smaller regions and, more importantly, this partition is not performed in advance but is dynamically and appropriately adjusted during the learning process. As a result, each smaller region varies in size and the degree of overlapping is also adjustable. This is in contrast to the localized storage scheme of the BOXES system [12], [27], in which the input space is divided into disjoint regions (boxes) and no generalization occurs beyond the confines of a given box. The fuzzification process in the RNN-FLCS also avoids the necessity of partitioning the input space into disjoint [12], [27] or overlapping [20], [30] small regions in advance. The drawback of this process is that it requires the learning control designer to know how to quantify or partition the input space properly according to the knowledge of the control task; otherwise, the input space can be easily partitioned in such a way that the controller will not work at all. In most cases, *a priori* selection of a sufficiently fine representation is required. However, a representation that is too fine may result in poor generalization capabilities and unnecessarily slow learning.

The second most important feature of the proposed RNN-FLCS is its dynamic structure/parameter learning ability and the rule combination process. The former can add new nodes to the RNN-FLCS properly, while the latter can combine some nodes with the same or a very similar control action. These functions realize the concept of "splitting" and "lumping" boxes in [31], where the authors tried to improve their BOXES system without much success. The third feature of the proposed RNN-FLCS is its stochastic search with multiparameter distribution functions. This results in the action network having a higher probability of finding a better action via the prediction signal from the evaluation network instead of performing random searches around the expected action using only single-parameter distribution functions. The fourth important feature of the proposed RNN-FLCS is the ease of incorporating expert knowledge into the fuzzy logic controller or the fuzzy predictor of the RNN-FLCS to greatly shorten the learning time. The continuous control scheme is another important feature of the RNN-FLCS. By performing fuzzy inference and defuzzification, the action network of the RNN-

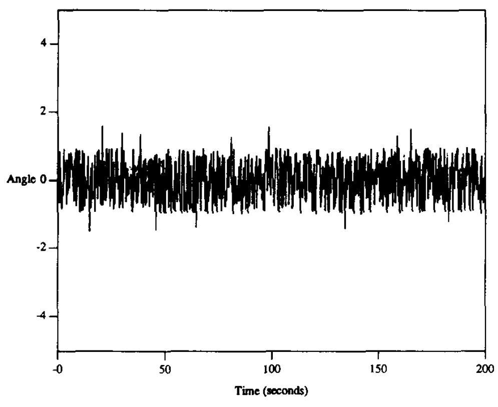


Fig. 9. Variations in θ produced by the learned RNN-FLCS.

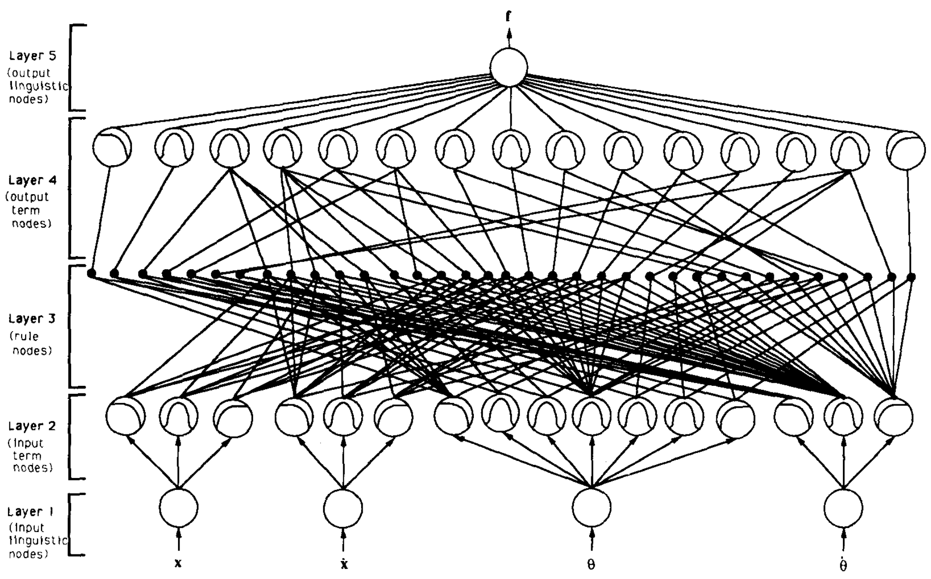

f
Layer 5 (output linguistic nodes)
Layer 4 (output term nodes)
Layer 3 (rule nodes)
Layer 2 (input term nodes)
Layer 1 (input linguistic nodes)
x
ẋ
θ
θ̇

Fig. 10. The learned fuzzy logic controller on the cart-pole balancing problem.

FLCS determines an analog output control signal. This is in contrast to the bang–bang control scheme used in [12], [27], [28], where the output can only have two values: $\pm 10N$. Thus, the angle deviation of the pole about the center point, as shown in Fig. 9, is rather small in our computer simulation.

Recently, Berenji and Khedkar [32], [33] independently proposed a FLC structure and its associated learning algorithm which are similar to what we propose here. They constructed a model, called Generalized Approximate Reasoning-based Intelligent Control (GARIC) architecture, for learning and tuning a fuzzy logic controller based on reinforcement signals from a dynamic system. Their architecture extends Anderson's method [28] by including *a priori* control knowledge of expert operators in terms of fuzzy control rules. In GARIC, a three-layer feedforward neural network, called Action Evaluation Network (AEN), was used to predict the external reinforcement signals and produce internal reinforcement signals. The role of the AEN can be considered to be parallel to our fuzzy predictor in the RNN-FLCS. They also adopted a five-layer feedforward neural network, called Action Selection Network (ASN), to implement a FLC as we did. Moreover, a stochastic action modifier was used to stochastically generate an actual action according to the expected action suggested by the ASN and the internal reinforcement signals from the AEN. The ASN, along with the stochastic action modifier, play the same role as the action network in our RNN-FLCS. Hence, there are two major differences between the structures of RNN-FLCS and GARIC. First, the RNN-FLCS uses a fuzzy predictor while the GARIC uses a general multilayer neural network for the reinforcement signal prediction. Second, in the RNN-FLCS, the action network (fuzzy controller) and the evaluation network (fuzzy predictor) share the same internal input representation. However, in GARIC, the corresponding two networks, the ASN and AEN, are structured independently. The learning algorithm in the AEN of GARIC used Sutton's AHC algorithm [12], [28] for the output units and the error backpropagation algorithm for the hidden units. The stochastic exploration technique and the error backpropagation algorithm were used to find the proper parameters in the ASN. This corresponds to the adjustment of the input/output membership functions of a fuzzy logic controller. Although these techniques are also adopted in the RNN-FLCS, our learning algorithms for the evaluation network and the action network are developed based on our previously proposed on-line structure/parameter learning algorithm [4]. Hence, in addition to the parameter learning, they can perform the structure learning and find the proper fuzzy logic rules dynamically. This can be considered the major difference between RNN-FLCS and GARIC. Furthermore, we have developed both the single-step and multistep fuzzy predictors to incorporate three different cases of reinforcement learning problems. This makes the RNN-FLCS a much more general structure than the GARIC. The learning speed of the GARIC for the cart-pole balancing problem was faster than that of the RNN-FLCS. However, a set of correct fuzzy logic rules must be given in advance by experts before initiating training of the GARIC.

VI. Conclusion

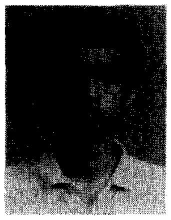
This paper described the development of integrating two NN-FLC's into an integrated Reinforcement Neural-Network-Based Fuzzy Logic Control System for solving various reinforcement learning problems. By combining the techniques of temporal difference, stochastic exploration, and the previously proposed on-line supervised structure/parameter learning algorithm, a reinforcement structure/parameter learning algorithm was derived for the RNN-FLCS. Using the proposed connectionist structure and learning algorithm, a fuzzy logic controller to control a plant and a fuzzy predictor to model the plant can be set up dynamically through simultaneous structure/parameter learning for various classes of reinforcement learning problems. The proposed RNN-FLCS 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 cart-pole balancing problem satisfactorily verified the validity and performance of the proposed reinforcement structure/parameter learning algorithm for RNN-FLCS. Future work will focus on an interesting problem of regulating the cart's position from an initial position to a set point, in addition to keeping the pole balance. This problem has multiple goals with different priorities. A hierarchical FLC design approach is needed. This is a topic of our ongoing research which will be discussed in a future paper.

Acknowledgment

The authors would like to thank the reviewers for their helpful suggestions in improving the quality of the final manuscript.

References

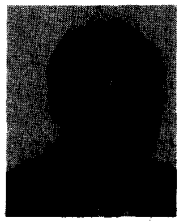
[1] 1. G. E. Hinton, "Connectionist learning procedures," *Art. Intell.*, vol. 40, no. 1, pp. 143–150, 1989.
[2] 2. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distrib. Proces.* Cambridge, MA: M.I.T. Press 1986, vol. 1, pp. 318–362.
[3] 3. C. T. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. C-40, no. 12, pp. 1320–1336, Dec. 1991.
[4] 4. C. T. Lin and C. S. G. Lee, "Real-time supervised structure/parameter learning for Fuzzy Neural Network," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, San Diego, CA, Mar. 1992, pp. 1283–1290.
[5] 5. A. Dickinson, *Contemporary Animal Learning Theory.* Cambridge, MA: Cambridge Univ. Press, 1980.
[6] 6. W. K. Estes, "Toward a statistical theory of learning," *Psych. Rev.*, vol. 57, pp. 94–107, 1950.
[7] 7. M. L. Tsetlin, *Automation Theory and Modeling of Biological Systems.* New York: Academic, 1973.
[8] 8. T. M. Cover and M. E. Hellman, "The two-armed bandit problem with time-invariant finite memory," *IEEE Trans. Inform. Theory,* vol. IT-16, pp. 185–195, 1970.
[9] 9. K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction.* Englewood Cliffs, NJ: Prentice Hall, 1989.
[10] 10. A. H. Klopf, *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Washington, DC: Hemisphere, 1982.
[11] 11. A. G. Barto and R. S. Sutton, "Landmark learning: an illustration of associative search," *Biol. Cybern.*, vol. 42, pp. 1–8, 1981.
[12] 12. A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-13, no. 5, pp. 834–847, 1983.
[13] 13. A. G. Barto and P. Anandan, "Pattern-recognizing stochastic learning automata," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-15, no. 3, pp. 360–375, 1985.
[14] 14. A. G. Barto and M. I. Jordan, "Gradient following without back-propagation in layered networks," in *Proc. 1987 Int. Joint Conf. Neural Netw.*, San Diego, CA, vol. II, pp. 629–636.
[15] 15. R. J. Williams, "A class of gradient-estimating algorithms for reinforcement learning in neural networks," in *Proc. 1987 Int. Joint Conf. Neural Netw.*, San Diego, CA, vol. II, pp. 601–608.
[16] 16. C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Parts I and II," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-20, no. 2, pp. 404–435, 1990.
[17] 17. L. A. Zadeh, "Fuzzy logic," *IEEE Comput.*, pp. 83–93, Apr. 1988.
[18] 18. J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation.* New York: Addison-Wesley, 1991, pp. 188–189.
[19] 19. J. A. Franklin, "Input space representation for reinforcement learning control," in *Proc. IEEE Int. Conf. Intell. Mach.*, 1989, pp. 115–122.
[20] 20. C. C. Lee and H. R. Berenji, "An intelligent controller based on approximate reasoning and reinforcement learning," in *Proc. IEEE Int. Conf. Intell. Mach.*, 1989, pp. 200–205.
[21] 21. R. S. Sutton, "Learning to predict by the methods of temporal difference," *Mach. Learning,* vol. 3, pp. 9–44, 1988.
[22] 22. P. J. Werbos, "A menu of design for reinforcement learning over time," in *Neural Networks for Control,* W. T. Miller, III, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: M.I.T. Press, 1990, ch. 3.
[23] 23. B. Widrow and M. E. Hoff, "Adaptive switching circuits," *1960 WESCON Convent. Rec.*, part IV, pp. 96–104.
[24] 24. R. H. Cannon, Jr., *Dynamics of Physical Systems.* New York: McGraw-Hill, 1967.
[25] 25. K. C. Cheok and N. K. Loh, "A ball-balancing demonstration of optimal and disturbance-accommodating control," *IEEE Contr. Syst. Mag.*, pp. 54–57, Feb. 1987.
[26] 26. B. Widrow, "The original adaptive neural net broom-balancer," in *Proc. Int. Symp. Circ. and Syst.*, May 1987, pp. 351–357.
[27] 27. D. Michie and R. A. Chambers, "BOXES: An experiment in adaptive control," in *Mach. Intell.*, E. Dale and D. Michie, Eds. Edinburgh, Scotland: Oliver and Boyd, 1968, vol. 2, pp. 137–152.
[28] 28. C. W. Anderson, "Strategy learning with multilayer connectionist representations," in *Proc. Fourth Int. Workshop on Mach. Learn.*, Irvine, CA, June 1987, pp. 103–114.
[29] 29. C. W. Anderson and W. T. Miller III, "Challenging control problems," in *Neural Networks for Control,* W. T. Miller, III, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: M.I.T. Press, 1990, ch. A.
[30] 30. J. S. Albus, "Mechanisms of planning and problem solving in the brain," *Math. Biosci.*, vol. 45, pp. 247–293, 1979.
[31] 31. D. Michie and R. A. Chambers, "\thinspace 'Boxes' as a model of pattern-formation," in *Towards a Theoretical Biology,* I. Prolegomena and C. H. Waddington, Eds. Edinburgh: Edinburgh Univ. Press, 1968, vol. 1, pp. 206–215.
[32] 32. H. R. Berenji, "An architecture for designing fuzzy controllers using neural networks," *Int. J. Approx. Reason.*, vol. 6, no. 2, pp. 267–292, Feb. 1992.
[33] 33. H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 724–740, 1992.



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

Since August 1992, he has been with the Department of Computer Information Science, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C. His current research interests are neural networks, fuzzy systems, learning systems, intelligent control, and artificial intelligence.

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



C. S. George Lee received the B.S. and M.S. degrees in electrical engineering from Washington State University in 1973 and 1974, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1978.

In 1978–1979, he taught at Purdue University and, in 1979–1985, at the University of Michigan. Since 1985, he has been with the School of Electrical Engineering, Purdue University, where he is currently Professor of Electrical Engineering. His current research interests include computational robotics, intelligent robotic assembly systems, and neural-network-based fuzzy logic control systems. He was an IEEE Computer Society Distinguished Visitor in 1983–1986, the Organizer and Chairman of the 1988 NATO Advanced Research Workshop on Sensor-Based Robots: Algorithms and Architectures, and the Secretary of the IEEE Robotics and Automation Society in 1988–1990. Currently, he is Vice-President for Technical Affairs and an AdCom member of the IEEE Robotics and Automation Society, and an Associate Editor of the *International Journal of Robotics and Automation.* He is a co-author of *Robotics: Control, Sensing, Vision, and Intelligence* (McGraw-Hill), and co-editor of *Tutorial on Robotics* (IEEE Computer Society Press).

Dr. Lee is a member of Sigma Xi and Tau Beta Pi.