Reinforcement Learning for An ART-Based Fuzzy Adaptive Learning Control Network

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

Abstract— **This paper proposes a reinforcement fuzzy adaptive learning control network (RFALCON) for solving various reinforcement learning problems. The proposed RFALCON is constructed by integrating two fuzzy adaptive learning control networks (FALCON's), each of which is a connectionist model with a feedforward multilayer network developed for the realization of a fuzzy controller. One FALCON performs as a critic network (fuzzy predictor), and the other as an action network (fuzzy controller). Using the temporal difference prediction method, the critic network can predict the external reinforcement signal and provide a more informative internal reinforcement signal to the action network. The action network performs a stochastic exploratory algorithm to adapt itself according to the internal reinforcement signal. An ART-based reinforcement structure/parameter-learning algorithm is developed for constructing the RFALCON dynamically. During the learning process, both structure learning and parameter learning are performed simultaneously in the two FALCON's. The proposed RFALCON can construct a fuzzy control system dynamically and automatically through a reward/penalty signal (i.e., a "good" or "bad" signal). It is best applied to the learning environment, where obtaining exact training data is expensive. The proposed RFALCON has two important features. First, it reduces the combinatorial demands placed by the standard methods for adaptive linearization of a system. Second, the RFALCON is a highly autonomous system. Initially, there are no hidden nodes (i.e., no membership functions or fuzzy rules). They are created and begin to grow as learning proceeds. The RFALCON can also dynamically partition the input–output spaces, tune activation (membership) functions, and find proper network connection types (fuzzy rules). Computer simulations have been conducted to illustrate the performance and applicability of the proposed learning scheme.**

I. Introduction

MOST of the supervised and unsupervised learning algorithms for neural networks require precise training data for setting the link weights and link connectivity of the nodes for various applications. For some real-world applications, precise training data 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 fuzzy adaptive learning control networks (FALCON's) [2] to the reinforcement learning problem.

Training data are very rough and coarse for the reinforcement learning problem, and they are only "evaluative" when compared with the "instructive" feedback in the supervised learning problem. Training a network with this kind of evaluative feedback is called reinforcement learning; this simple evaluative feedback is scalar and is called the reinforcement signal. In addition to the roughness and noninstructive nature of the reinforcement signal, a more challenging problem to reinforcement learning is that a reinforcement signal may only be available at a time long after a sequence of actions has occurred; this signal may be caused by an action several time steps before or by the whole sequence of actions with varying degrees of contribution. In the latter case, it is difficult to determine which action to reward or punish. For example, in chess, the final result (win or lose) cannot be known until after a long sequence of moves. This is the well-known credit assignment problem in artificial intelligence [3]. To solve the long time-delay problem, reinforcement learning systems need the capability to predict. Reinforcement learning that is able to predict is also much more useful than supervised learning schemes in dynamic control problems, since the success or failure signal might be known only 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 [4] than is supervised learning. Research shows that animals and most cells adapt themselves to environments according to reinforcement signals from the external world or other cells. Moreover, animals can learn extensively about their environments even before the external reinforcement signal as introduced in [5]. This situation is very similar to learning many high-level intelligent actions such as how to drive a car.

The development of reinforcement learning can be roughly divided into two stages. The first 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 were making independent efforts in 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]. 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]. More details on stochastic learning automata can be found in [9]. Representative of the second stage development of reinforcement learning is the associative

Manuscript received May 26, 1994; revised January 29, 1995 and May 26, 1995. This work was supported by the National Science Council, Republic of China, under Grant NSC 83-0422-E-009-004.

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

Publisher Item Identifier S 1045-9227(96)01228-3.

reinforcement learning in which people try to 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]. The idea of their proposed architecture, called the actor-critic (adaptive heuristic critic) architecture, was fully developed in [13]. They also proposed the associative reward-penalty (A_{R-P}) algorithm for adaptive elements called A_{R-P} elements [14], and several generalizations of A_{R-P} algorithm have been proposed [15]. Williams formulated the reinforcement learning problem as a gradient-following procedure [17], and he identified a class of algorithms, called REINFORCE algorithms, that possess the gradient ascent property. These algorithms, however, still do not include the full A_{R-P} algorithms. Recently, Berenji and Khedkar [18] proposed a fuzzy logic controller and its associated learning algorithm. Their architecture extends Anderson's method [19] by including *a priori* control knowledge of expert operators in terms of fuzzy control rules. Lin and Lee [20] also proposed a reinforcement neural-network-based fuzzy logic control system (RNN-FLCS) for solving various reinforcement learning problems. The RNN-FLCS can find proper network structure and parameters simultaneously and dynamically.

In this paper, we shall apply the technique of associative reinforcement learning to our proposed reinforcement FALCON system. The proposed learning system can construct a fuzzy controller automatically and dynamically by means of a reward-penalty (i.e., good–bad) signal or from very simple fuzzy feedback information such as "high," "too high," "low," and "too low." Moreover, there is a possibility of a long 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 fuzzy adaptive learning control network (RFALCON) is proposed, which consists of two closely integrated FALCON's. The FALCON is a feedforward multilayer network that integrates the basic elements and functions of a traditional fuzzy controller into a connectionist structure. In this connectionist structure, the input and output nodes represent the input states and output control/decision signals, respectively, and, in the hidden layers, there are node functions as membership functions (activation functions) and fuzzy logic rules (connection types). In the RFALCON, the FALCON used for the action network (fuzzy controller) can choose a proper action according to the current input vector. Its functions like the one proposed in [2] except that there is no "teacher" to indicate output errors for the action network to learn in the reinforcement learning problem. The other FALCON is used as the critic network (fuzzy predictor), and provides the action network more informative and earlier internal reinforcement signals from which to learn.

Associated with the proposed RFALCON is an ART-based reinforcement structure/parameter-learning algorithm. This algorithm uses the temporal difference technique on the critic network to determine the output errors for multistep prediction. With this knowledge of output errors, the ART-based on-line

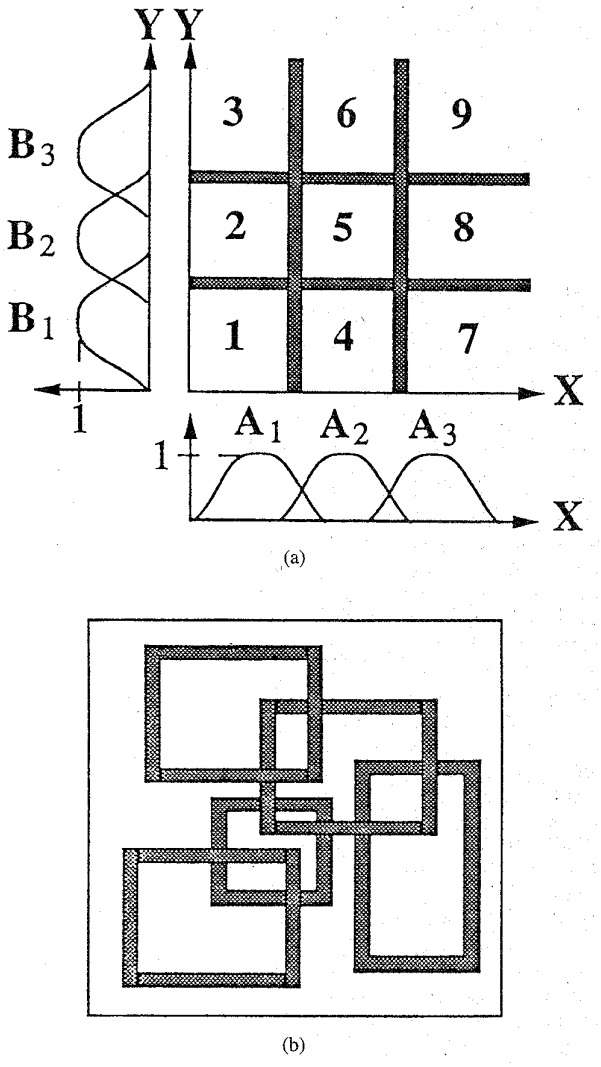


Fig. 1. (a) Grid-type fuzzy partitioning. (b) Flexible hyperbox fuzzy partitioning.

supervised structure/parameter-learning algorithm developed in [2] can train the critic network to do fuzzy clustering in the input–output spaces and find proper fuzzy logic rules (connection types) dynamically by associating input clusters and output clusters. For the action network, the reinforcement structure/parameter-learning algorithm allows its output nodes to perform stochastic exploration on the output space. With the internal reinforcement signals from the critic network, 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 supervised learning algorithm described in [2]. Thus, the proposed reinforcement structure/parameter-learning algorithm is basically composed of the temporal difference techniques, stochastic exploration, and on-line supervised structure/parameter-learning algorithm presented in [2]. It can thus on-line partition the input–output spaces, tune activation (membership) functions, and find proper network

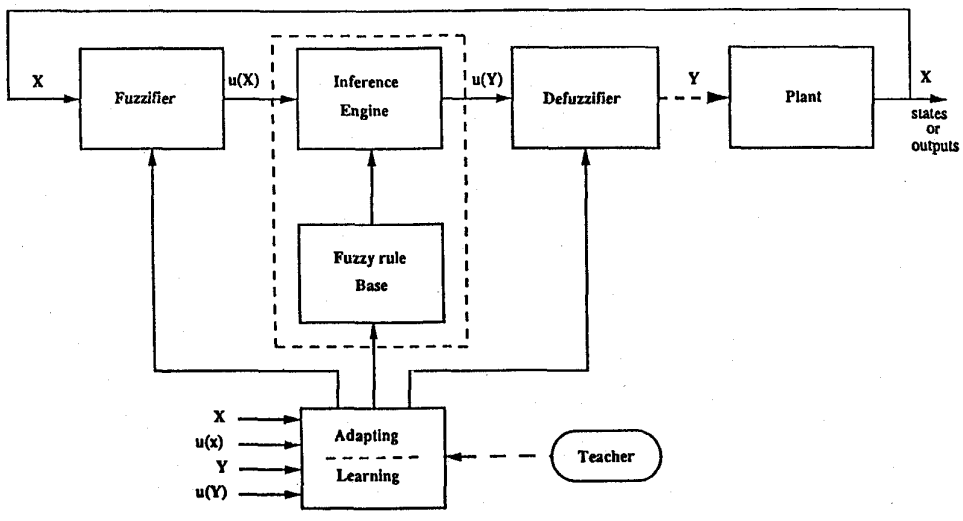


Fig. 2. Functional diagram of a fuzzy controller.

connection types (fuzzy logic rules) for an RFALCON dynamically by means of an external reinforcement signal. Moreover, learning can proceed even before an external reinforcement feedback is available.

An important feature of the proposed learning system is that it can dynamically partition the input-state space and output-control space using irregular fuzzy hyperboxes according to the distribution of environment states and reinforcement signals. In many existing fuzzy or neural fuzzy control systems, including those introduced in the preceding paragraphs, the input and output spaces are always partitioned into "grids," each defining a fuzzy region. The overlapping regions between the grids provide a smooth and continuous membership output surface. Consider, for example, a fuzzy logic controller with two input variables. If each of them contains three fuzzy terms (e.g., "small," "medium," and "large,") then the corresponding input-space partition is as shown in Fig. 1(a). Although during the learning process, the position and shape of membership functions will be changed, they are still inherently grid-type partitions. Due to its simplicity and intuitiveness, grid-type partitioning of input and output spaces has been used in the design of many existing fuzzy systems. As the number of input–output variables increases, however, the number of partitioned grids grows combinatorially. As a result, the memory or hardware size required may become impracticably large. This results in more learning difficulty since with finer space partitioning, more training samples are needed or insufficient learning takes place. To avoid combinatorial growth of partitioned grids in complex systems, more flexible and irregular space partitioning methods must be developed. Fig. 1(b) shows a proposed partitioning method in the RFALCON system. The problem of space partitioning from numerical training data is basically a clustering problem. The proposed system applies the fuzzy adaptive resonance theory (fuzzy ART) proposed by Carpenter *et al.* [24], [25] to do fuzzy clustering in the input–output spaces and find proper fuzzy logic rules dynamically by associating input clusters with output clusters. The backpropagation learning scheme is then used for tuning input–output membership functions. Hence, in the RFALCON, the fuzzy ART is used for structure learning and the backpropagation algorithm for parameter learning. The RFALCON can thus on-line partition the input–output spaces, tune membership functions, and find proper fuzzy logic rules dynamically on the fly. More notably, in this learning scheme, only the reinforcement signals from the outside world need to be provided. Users do not need to give the initial fuzzy partitions, membership functions, or fuzzy logic rules, hence, there are no hidden nodes at the beginning of learning; they are created and begin to grow as the first reinforcement signal arrives.

This paper is organized as follows: Section II describes the basic structure and functions of our previously proposed FALCON [2]. The proposed RFALCON and the corresponding reinforcement structure/parameter-learning algorithm with multistep prediction capability are presented in Section III. In Section IV, the cart-pole balancing system and the ball and beam system are simulated to demonstrate the ability of the proposed RFALCON. Performance comparison with some other existing systems is described in Section V. Finally, conclusions are summarized in Section VI.

II. The Structure of the FALCON

We shall briefly introduce basic components of conventional fuzzy control systems (for more details, please refer to [26] and [27]), and then propose our connectionist model.

A. Basic Structure of A Fuzzy Control System

Fig. 2 shows the basic structure of a conventional fuzzy control system with a learning/adapting component. Before proceeding, we must define some important terms. A fuzzy set F in a universe of discourse U is characterized by a membership function $\mu_F : U \rightarrow [0,1]$. Thus, a fuzzy set F in U may be represented as a set of ordered pairs. Each pair consists of a generic element u and its grade of membership function $\mu_F(u)$; that is, $F = \{(u, \mu_F(u)) \mid u \in U\}$. u is called a support value if $\mu_F(u) > 0$. If U is a continuous universe and

F is normal and convex (i.e., $\max_{u \in U} \mu_F(u) = 1$ and $\mu_F(\lambda u_1 + (1-\lambda)u_2) \geq \min(\mu_F(u_1), \mu_F(u_2))$, $u_1, u_2 \in U, \lambda \in [0,1]$), then F is a fuzzy number. A linguistic variable x in a universe of discourse U is characterized by $T(x) = \{T_x^1, T_x^2, \cdots, T_x^k\}$ and $M(x) = \{M_x^1, M_x^2, \cdots, M_x^k\}$, where $T(x)$ is the term set of x; that is, the set of names of linguistic values of x with each value T_x^i being a fuzzy number with membership function M_x^i defined on U. So $M(x)$ is a semantic rule for associating with each value its meaning. For example, if x indicates speed, then $T(x)$ may be {slow, medium, fast}. Following the above definition, the input vector X which includes the input state linguistic variables x_i's, and the output state vector Y which includes the output state linguistic variables y_i's in Fig. 2 can be defined as

$$X = \left\{(x_i, U_i, \{T_{x_i}^1, T_{x_i}^2, \cdots, T_{x_i}^{k_i}\}, \{M_{x_i}^1, M_{x_i}^2, \cdots, M_{x_i}^{k_i}\}) \mid_{i=1,\cdots,n}\right\} \tag{1}$$

$$Y = \left\{(y_i, U_i', \{T_{y_i}^1, T_{y_i}^2, \cdots, T_{y_i}^{l_i}\}, \{M_{y_i}^1, M_{y_i}^2, \cdots, M_{y_i}^{l_i}\}) \mid_{i=1,\cdots,m}\right\}. \tag{2}$$

The fuzzifier in Fig. 2 is a mapping from an observed input space to fuzzy sets in certain input universe of discourse. So a specific value $x_i(t)$ at time t is mapped to the fuzzy set $T_{x_i}^1$ with degree $M_{x_i}^1(x_i(t))$ and to the fuzzy set $T_{x_i}^2$ with degree $M_{x_i}^2(x_i(t))$, and so on.

The fuzzy rule base in Fig. 2 contains a set of fuzzy logic rules R. For a multi-input and multioutput (MIMO) system, we have

$$R = \{R_{MIMO}^1, R_{MIMO}^2, \cdots, R_{MIMO}^n\}$$

where the ith fuzzy logic rule is

$$R_{MIMO}^i : \text{IF } (x_1 \text{ is } T_{x_1} \text{ and } \cdots \text{ and } x_p \text{ is } T_{x_p}) \text{ THEN } (y_1 \text{ is } T_{y_1} \text{ and } \cdots \text{ and } y_q \text{ is } T_{y_q}).$$

The preconditions of R_{MIMO}^i form a fuzzy set $T_{x_1} \times \cdots \times T_{x_p}$ and the consequent of R_{MIMO}^i is the union of q independent outputs. So the rule can be represented by a fuzzy implication

$$R_{MIMO}^i : (T_{x_1} \times \cdots \times T_{x_p}) \rightarrow (T_{y_1} + \cdots + T_{y_q})$$

where "+" represents the union of independent variables. Since the outputs of MIMO rule are independent, the general rule structure of MIMO fuzzy system can be represented as a collection of multi-input and single-output (MISO) fuzzy systems by decomposing the above rule into q subrules with T_{y_i} as the single consequent of the ith subrule. For clarity, we shall consider MISO system in the following analysis. A sample rule follows:

IF the speed is too slow and the acceleration is decreasing, THEN increase power strongly.

The inference engine in Fig. 2 matches the rule preconditions in the fuzzy rule base with the input state linguistic terms and performs implication. For example, if there were two rules:

R1: IF x_1 is $T_{x_1}^1$ and x_2 is $T_{x_2}^1$, THEN y is T_y^1.

R2: IF x_1 is $T_{x_1}^2$ and x_2 is $T_{x_2}^2$, THEN y is T_y^2.

Then the firing strengths of rules R1 and R2 are defined as α_1 and α_2, respectively. Here α_i is defined as

$$\alpha_i = M_{x_1}^i(x_1) \wedge M_{x_2}^i(x_2) \tag{3}$$

where "$\wedge$" is the fuzzy AND operation. The most commonly used fuzzy AND operations are intersection and algebraic product [26], [27].

Rules R1 and R2 lead to the corresponding decision with the membership function, $\hat{M}_y^i(w)$, $i = 1, 2$, which is defined as

$$\hat{M}_y^i(w) = \alpha_i \wedge M_y^i(w) \tag{4}$$

where w is the variable that represents the membership function support values. Combining these decisions, we obtain the output decision

$$\hat{M}_y(w) = \hat{M}_y^1(w) \vee \hat{M}_y^2(w) \tag{5}$$

where "$\vee$" is the fuzzy OR operation. The most commonly used fuzzy OR operations are union and bounded sum [26], [27].

Notice that the last result is a membership function curve. Before feeding the signal to the plant, we must defuzzify it to get a crisp decision, which is what defuzzifier block in Fig. 2 does. Among commonly used defuzzification strategies, the center of area method yields a superior result [26], [27]. Let w_j be the support value at which the membership function, $\hat{M}_y^j(w)$, reaches the maximum value $\hat{M}_y^j(w)|_{w=w_j}$. Then the defuzzification output is

$$y = \frac{\sum_j \hat{M}_y^j(w_j) w_j}{\sum_j \hat{M}_y^j(w_j)}. \tag{6}$$

The preceding describes the standard function operations in a conventional fuzzy control system, although there are some alternatives for fuzzy OR, fuzzy AND, and reasoning operations [26], [27].

Most system designers usually chose their membership functions empirically and constructed fuzzy logical rules supplied by experts. Enabling fuzzy control systems to learn is an important issue. The learning/adapting block in Fig. 2 represents this function. This learning/adapting block finds suitable fuzzy logic rules and adapts the fuzzifier and the defuzzifier to find the proper shapes and membership function overlaps by learning the desired output, or in response to external reinforcement signals. The aim of this paper is to present an adaptive fuzzy control system design that can adapt itself in response to external reinforcement signals. In the next section, the FALCON, a connectionist model, is proposed as just such a fuzzy control system. This neural-network-based architecture eliminates the rule-matching process and distributively stores control knowledge in the connection types and link weights. More importantly, the connectionist architecture is a natural structure with which to perform neural-network-based learning.

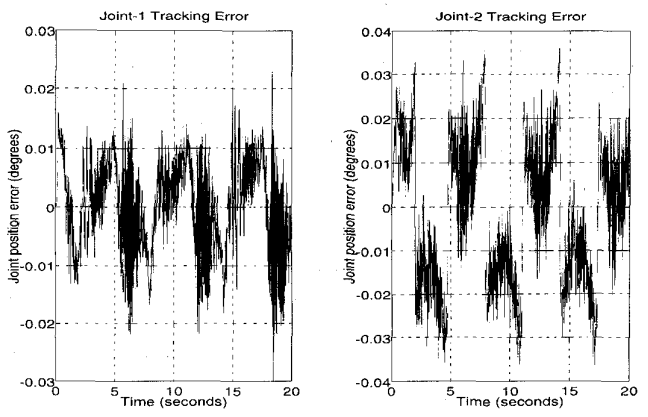


Fig. 3. The proposed FALCON.

B. A Connectionist Fuzzy Control System

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

The FALCON uses the technique of complement coding from Fuzzy ART [24] to normalize the input–output training vectors. Complement coding is a normalization process that rescales an n-dimensional vector in $\Re^n, \mathbf{x} = (x_1, x_2, \cdots, x_n)$, to its $2n$-dimensional complement coding form in $[0,1]^{2n}, \mathbf{x}'$, such that

$$\begin{aligned}\mathbf{x}' &\equiv (\bar{x}_1, \bar{x}_1^c, \bar{x}_2, \bar{x}_2^c, \cdots, \bar{x}_n, \bar{x}_n^c) \\ &= (\bar{x}_1, 1-\bar{x}_1, \bar{x}_2, 1-\bar{x}_2, \cdots, \bar{x}_n, 1-\bar{x}_n)\end{aligned} \tag{7}$$

where $(\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n) = \bar{\mathbf{x}} = \mathbf{x}/\|\mathbf{x}\|$ and $\bar{x}_1^c$ is the complement of $\bar{x}_1$, i.e., $\bar{x}_1^c = 1 - \bar{x}_1$. As mentioned in [24], complement coding helps avoid the problem of category proliferation when using fuzzy ART for fuzzy clustering. It also preserves training vector amplitude information. In applying the complement coding technique to the FALCON, all training vectors (either input state vectors or desired output vectors) are transformed to their complement coded forms in the preprocessing process, and the transformed vectors then used for training.

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

$$\text{net-input} = f(z_1^{(k)}, z_2^{(k)}, \cdots, z_p^{(k)}; w_1^{(k)}, w_2^{(k)}, \cdots, w_p^{(k)}) \tag{8}$$

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

$$\text{output} = a(f) \tag{9}$$

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

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

$$f(\bar{x}_i, \bar{x}_i^c) = (\bar{x}_i, \bar{x}_i^c) = (\bar{x}_i, 1-\bar{x}_i) \text{ and } a(f) = f. \tag{10}$$

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

Layer 2: Nodes in this layer are called input term nodes and each represents a term of an input linguistic variable, and acts as a one-dimensional membership function. The following trapezoidal membership function [31] is used:

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

and

$$a(f) = f \tag{11}$$

where $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ are, respectively, the left-flat and right-flat points of the trapezoidal membership function of the jth input term node of the ith input linguistic node [see Fig. 4(a)]; $z_{ij}^{(2)}$ is the input to the jth input term node from the ith input linguistic node (i.e., $z_{ij}^{(2)} = \bar{x}_i$); and

$$g(s,\gamma) = \begin{cases} 1 & \text{if } s\gamma > 1 \\ s\gamma & \text{if } 0 \le s\gamma \le 1 \\ 0 & \text{if } s\gamma < 0. \end{cases} \tag{12}$$

The parameter γ is the sensitivity parameter that regulates the fuzziness of the trapezoidal membership function. A large γ means a more crisp fuzzy set, and a smaller γ makes the fuzzy set less crisp. A set of n input term nodes (one for each

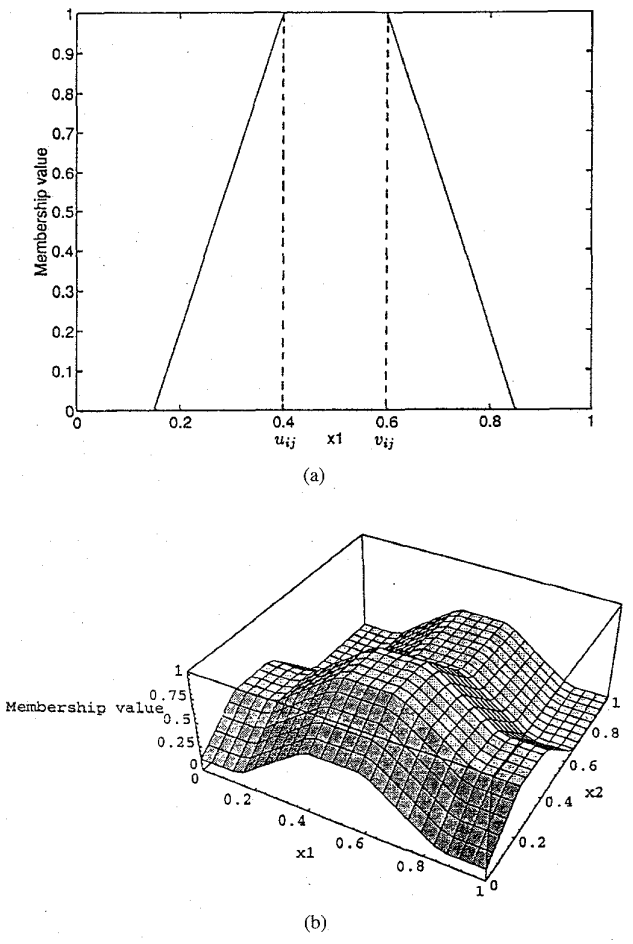


Fig. 4. (a) One-dimensional trapezoidal membership function. (b) Two-dimensional trapezoidal membership function.

input linguistic node) is connected to a rule node in layer 3 where its outputs are combined. This defines an n-dimensional membership function in the input space, with each dimension specified by one input term node in the set. Hence, each input linguistic node has the same number of term nodes. That is, each input linguistic variable has the same number of terms in the FALCON. This is also true for output linguistic nodes. A layer-2 link connects an input linguistic node to one of its term nodes. There are two weights on each layer-2 link. We denote the two weights on the link from input node i (corresponding to the input linguistic variable x_i) to its jth term node as $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ (see Fig 3). These two weights define the membership function in (11). The two weights, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, correspond respectively to the two inputs, $\bar{x}_i$ and $\bar{x}_i^c$ from the input linguistic node i. More precisely, $\bar{x}_i$ and $\bar{x}_i^c$, the two inputs to the input term node j, will be used during the fuzzy-ART clustering process in FALCON's structure-learning step to decide $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, respectively. In FALCON's parameter-learning step and normal operating, only $\bar{x}_i$ is used in the forward reasoning process [i.e., $z_{ij}^{(2)} = \bar{x}_i$ in (11)]. We detail the FALCON learning scheme in Section III.

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

$$f(z_i^{(3)}) = \sum_{i=1}^{n} z_i^{(3)} \text{ and } a(f) = f \tag{13}$$

where $z_i^{(3)}$ is the ith input to a node in layer 3 and the summation is over the inputs of this node. The link weight in layer 3 ($w_i^{(3)}$) is then unity. Note that the summation in the above equation is equivalent to defining a multidimensional (n-dimensional) membership function, which is the summation of the trapezoid functions in (11) over i. This forms a multidimensional trapezoidal membership function called the hyperbox membership function [31], since it is defined on a hyperbox in the input space. The corners of the hyperbox are decided by the layer-2 weights, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$, for all i's. More clearly, the interval $[u_{ij}^{(2)}, v_{ij}^{(2)}]$ defines the edge of the hyperbox in the ith dimension. Hence, the weight vector, $[(u_{1j}^{(2)}, v_{1j}^{(2)}), \cdots, (u_{ij}^{(2)}, v_{ij}^{(2)}), \cdots, (u_{nj}^{(2)}, v_{nj}^{(2)})]$, defines a hyperbox in the input space. An illustration of a two-dimensional hyperbox membership function is shown in Fig. 4(b). The rule nodes output are connected to sets of m output term nodes in layer 4, one for each output linguistic variable. This set of output term nodes defines an m-dimensional trapezoidal (hyperbox) membership function in the output space that specifies the consequent of the rule node. Note that different rule nodes may be connected to the same output hyperbox (i.e., they may have the same consequent), as is shown in Fig. 3.

Layer 4: The nodes in this layer are called output term nodes; each has two operating modes: down–up transmission and up–down transmission modes (see Fig. 3). In down–up transmission mode, the links in layer 4 perform the fuzzy OR operation on fired (activated) rule nodes that have the same consequent

$$f(z_i^{(4)}) = \max(z_1^{(4)}, z_2^{(4)}, \cdots, z_p^{(4)}) \text{ and } a(f) = f \tag{14}$$

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

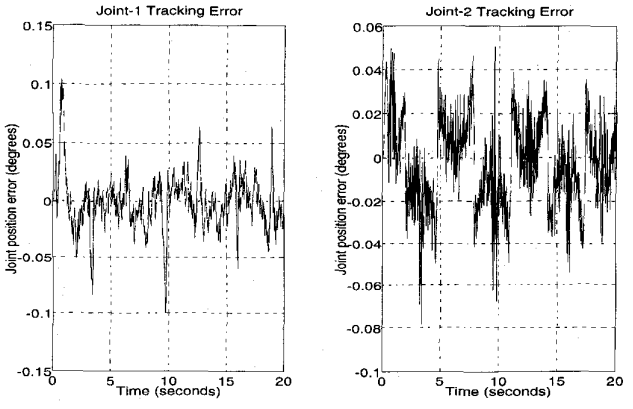


Fig. 5. The fuzzy reasoning process in the FALCON model.

functions) in the output space. More clearly, the weight vector, $[(u_{1j}^{(5)}, v_{1j}^{(5)}), \cdots, (u_{ij}^{(5)}, v_{ij}^{(5)}), \cdots, (u_{mj}^{(5)}, v_{mj}^{(5)})]$, defines a hyperbox in the output space.

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

$$f(\bar{y}_i, \bar{y}_i^c) = (\bar{y}_i, \bar{y}_i^c) = (\bar{y}_i, 1 - \bar{y}_i) \text{ and } a(f) = f \qquad (15)$$

where $\bar{y}_i$ is the ith element of the normalized desired output vector. Notice that complement coding is also performed on the desired output vectors. Thus, as mentioned above, there are two weights on each of the up-down transmission links in layer 5 (the $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ shown in Fig. 3). The weights define hyperboxes and the associated hyperbox membership functions in the output space. The second kind of node performs down-up transmission for decision signal output. These nodes and the layer-5 down-up transmission links attached to them act as a defuzzifier. If $u_{ij}^{(5)}$ and $v_{ij}^{(5)}$ are the corners of the hyperbox of the jth term of the ith output linguistic variable y_i, then the following functions can be used to simulate the center of area defuzzification method:

$$f(z_j^{(5)}) = \sum_j w_{ij}^{(5)} z_j^{(5)} = \sum_j m_{ij}^{(5)} z_j^{(5)}$$
$$\text{and } a(f) = \frac{f}{\sum_j z_j^{(5)}} \qquad (16)$$

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

The fuzzy reasoning process in the FALCON is illustrated in Fig. 5, which shows a graphic interpretation of the center of area defuzzification method. Here, we consider a two-input and two-output case. As shown in the figure, three hyperboxes (IH1, IH2, and IH3) are formed in the input space and two hyperboxes (OH1, OH2) are formed in the output space. These hyperboxes are defined by the weights u_{ij}, v_{ij}, u_{ij}', and v_{ij}'. The three fuzzy rules indicated in the figure are "IF $\mathbf{x}$ is IH1 THEN $\mathbf{y}$ is OH1 (rule 1)," "IF $\mathbf{x}$ is IH2 THEN $\mathbf{y}$ is OH1 (rule 2)," and "IF $\mathbf{x}$ is IH3 THEN $\mathbf{y}$ is OH2 (rule 3)," where $\mathbf{x} = (x_1, x_2)$ and $\mathbf{y} = (y_1, y_2)$. If an input pattern is located inside a hyperbox, the membership value is equal to one [see (12)]. In this figure, according to (14), z_1 is obtained by performing fuzzy OR (maximum) operation on the inferred results of rules 1 and 2, which have the same consequent, OH1. Also according to (14), z_2 is directly the inferred result of rule 3. z_1 and z_2 are then defuzzified to get the final output according to (16).

Note that with the proposed learning algorithms developed in Section III, no input–output term nodes and no rule node are presented when learning begins. They are created dynamically as on-line teaching signals are received and learning proceeds. In other words, the FALCON initially has only input and output linguistic nodes before training, other input and output term nodes and rule nodes are added during the learning process.

An on-line supervised learning algorithm originally proposed in [2] is associated with the FALCON to perform on-line training. This algorithm works very well when supervised training data are available on-line, however, it requires precise training data to indicate the exact desired outputs, and 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 right–wrong decision from the current controller. Thus, we propose two FALCON's integrated into a structure called RFALCON to train networks with this kind of evaluative feedback. The RFALCON and an associated reinforcement learning algorithm are presented in the next section.

III. Learning Algorithm for the RFALCON with a Multistep Critic Network

This section presents details of the integrated RFALCON network and associated learning algorithm we propose. The RFALCON consists of two FALCON's, one of which is used as an action network, and the other as a critic network (see Fig. 6). The reinforcement learning algorithm combines structure learning and parameter learning to determine the proper corners of the hyperbox (u_{ij}'s and v_{ij}'s) for each term node in layers 2 and 4. It also learns fuzzy logic rules and link connection types in layers 3 and 4; that is, the precondition and consequent links of the rule nodes. All the learning is performed on both FALCON's simultaneously and only conducted by a reinforcement signal feedback from the external environment. Unlike the supervised learning problem

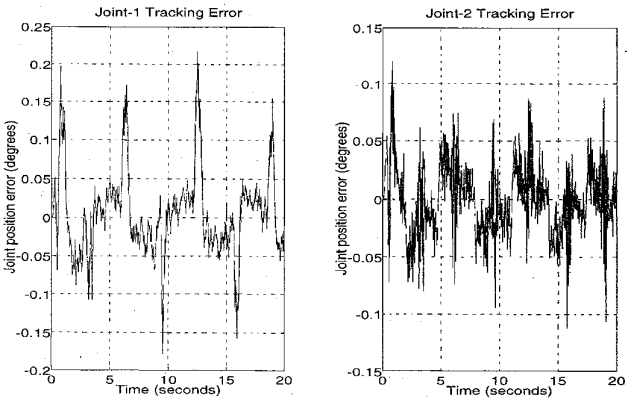


Fig. 6. The proposed RFALCON.

in which the correct "target" output values are given for each input pattern to instruct network learning, the reinforcement learning problem has only very simple "evaluative" or "critic" information available for learning rather than "instructive" information. In the extreme case, there is only a single bit of information to indicate whether the output is right or wrong. Fig. 6 shows how a network and its training environment interact in a reinforcement learning problem. The environment supplies a time-varying input vector to the network, receives its time-varying output-action vectors, and then provides a time-varying scalar reinforcement signal. In this paper, the reinforcement signal $r(t)$ is two-valued, $r(t) \in \{-1, 0\}$, such that $r(t) = 0$ means "a success" and $r(t) = -1$ means "a failure." We also assume that $r(t)$ is the reinforcement signal available at time step t and is caused by the inputs and actions chosen at earlier time steps (i.e., at time steps $t-1, t-2, \cdots$). The goal 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 external reinforcement signal is highly dependent 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 environment states and network outputs. In most cases, the environment is itself governed by a complicated dynamic process, in which reinforcement signals and input patterns may depend on past network outputs. In a chess game, for example, 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 this class of reinforcement learning problems, a system, called the RFALCON, is proposed. Associated with the RFALCON is a reinforcement structure/parameter-learning algorithm. As Fig. 6 shows, the proposed RFALCON consists of two FALCON's: one FALCON for the action network (fuzzy controller) and the other FALCON for the critic network (fuzzy predictor). Each network has exactly the same structure as shown in Fig. 3. The action network can have multiple outputs as shown in Fig. 3, although only one output node is shown in Fig. 6. In the multioutput case, all the output nodes of the action network receive the same internal reinforcement signals from the critic network, which has only one output node since it is used to predict the external scalar reinforcement signal. Since we want to solve reinforcement learning problems in which the external reinforcement signal is available only after a long sequence of actions have been passed on to the environment, we need a multistep critic network to predict the external 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 and predicted reinforcement signals. The critic network models the environment such that it can perform a multistep prediction of the reinforcement signal that will eventually be obtained from the environment for the current action chosen by the action network. The multistep predicted reinforcement signal thus enables both the action network and the critic network to learn at each time step without waiting for the arrival of an external reinforcement signal, greatly accelerating the learning of both networks.

In this section, we introduce the proposed reinforcement structure/parameter-learning algorithm for the RFALCON with a multistep critic network. We first consider the reinforcement structure/parameter-learning scheme for the multistep critic network and then introduce the reinforcement learning scheme for the action network.

A. The Multistep Critic Network

We shall use a FALCON to develop a multistep critic network that can perform multistep prediction of an external reinforcement signal. When both the reinforcement signal and input patterns from the environment depend arbitrarily on the past history of the action network outputs and the action network only receives 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 long sequences leading up to eventual successes or failures. Thus, to handle this class of reinforcement learning problem, we need to solve the temporal credit assignment problem along with solving the original structural credit assignment problem concerning attribution of network errors to different connections or weights. The solution to the temporal credit assignment problem we propose in the RFALCON is including a multistep critic network that predicts the reinforcement signal at each time step in a given period without any external reinforcement signals from the environment. This will ensure that both the critic network and the action network can both update their parameters and structures during the period without any evaluative feedback from the environment. To solve the temporal credit assignment problem, we used a technique based on the temporal-difference method, which is often closely associated with dynamic programming techniques [12], [16], [29]. Unlike the single-step prediction and the supervised learning methods which assign credit according to the difference between the predicted and actual outputs, the temporal-difference methods assign credit according to the difference between temporally successive predictions. Notice that the term "multistep prediction" used here means the critic network can predict a value that will

be available several time steps later, although it does such prediction at each time step to improve its prediction accuracy.

In this paper, we consider the case of infinitely discounted predictions [16], [20]. In this case, the critic network output, p_{t-1}, predicts accumulatively discounted outcomes (external reinforcement signals) $\sum_{k=0}^{\infty} \xi^k r_{t+k} = r_t + \xi p_t$, where p_{t-1} is the output of the critic network at time $t-1$, r_t is the actual outcome occurring between time steps $t-1$ and t, and the discount-rate parameter ξ, $0 \leq \xi < 1$, determines the extent to which we are concerned with short- or long-range predictions. Since the actual outcome (external reinforcement signal), r_t, is not available at each time step in the problems we consider, r_t is set to zero during the period when there is no external reinforcement signal. The actual external reinforcement signal may be available at some unexpected time long after time step $t-1$. The infinitely discounted prediction is used for prediction problems in which exact success or failure may never be completely known. In this case, the prediction error is $(r_t + \xi p_t) - p_{t-1}$, and the learning rule is

$$\Delta w_t = \eta(r_t + \xi p_t - p_{t-1}) \sum_{k=1}^{t-1} \lambda^{t-k-1} \nabla_w p_k \qquad (17)$$

where w represents the adjustable network parameter (weight), η is the learning rate, and λ is the recency weighting factor with which weight update due to the predictions of observation vectors occurring k steps in the past are weighted by λ^k. In applying the temporal difference procedures to the proposed RFALCON, we let $\lambda = 0$ for efficiency and accuracy [16]. A general learning rule used for infinite discounted predictions is thus

$$\Delta w_t = \eta(r_t + \xi p_t - p_{t-1}) \nabla_w p_{t-1}. \qquad (18)$$

We shall next derive the learning rule for the multistep critic network according to (18). Here, $p(t) \equiv p_t$ is the single output of the critic network for the network's current parameter, $w(t)$, and current given input state vector, $x(t)$, at time step t. According to (18), let

$$\hat{r}(t) = r(t) + \xi p(t) - p(t-1), 0 \leq \xi < 1. \qquad (19)$$

Then $\hat{r}(t)$ is the multistep critic network's output node error signal. With the (predicted) desired output, $p(t)$, and the output error signal, $\hat{r}(t)$, available, we can use the supervised learning techniques to train the critic network. Note that if the environment provides a reinforcement signal at each time step, the critic network can "single-step," or calculate the actual learning output error at each time step. Thus, the critic network can operate as either a multistep or single-step predictor.

Thus far, we have formulated the multistep prediction problem as a supervised learning problem, so the on-line learning algorithm proposed in [2] can be adopted directly here. This learning scheme uses the fast-learn fuzzy ART for structure learning and the backpropagation algorithm for parameter learning of each incoming training pattern. When learning begins, only input and output linguistic nodes are presented, and user need not provide fuzzy partitions, membership functions, and fuzzy logic rules. As training data are received, input and output term nodes, and rule nodes are created dynamically. Then as learning proceeds, more are added as they are needed.

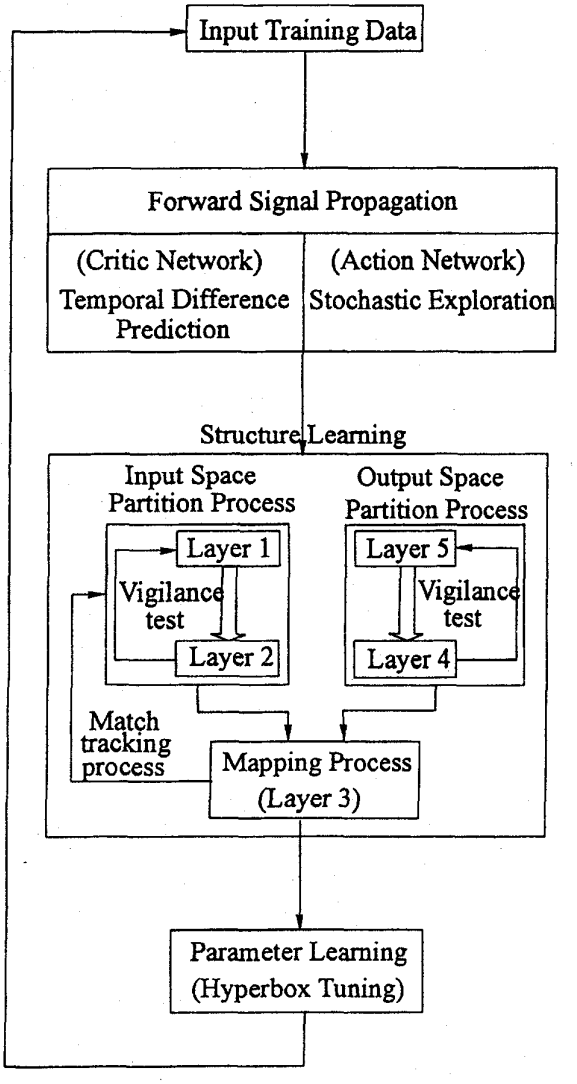


Fig. 7. Flowchart of the proposed reinforcement structure/parameter-learning algorithm.

We present the reinforcement learning algorithm for the critic network in detail in the rest of this section. A flowchart of this reinforcement learning algorithm is shown in Fig. 7. Basically, we use the temporal difference prediction method to find the output error of the critic network. The error is then used for training the critic network by the proposed reinforcement learning algorithm. This learning algorithm consists of two steps: the structure-learning step and the parameter-learning step, as introduced in the following two sections, respectively.

1) The Structure-Learning Step: The structure-learning task can be stated as: Given input training data at time t, $x_i(t), i = 1, \cdots, n$ and desired output value $r(t)$, we need proper fuzzy partitions, membership functions, and fuzzy logic rules. At this stage, the network works in a two-sided manner; that is, the

nodes and links in layer 4 are in the up-down transmission mode so training input and output data are fed in from both sides.

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

a) Input Fuzzy Clustering Process: We use the fuzzy ART fast learning algorithm [24,25] to find the input membership function parameters, $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$. This is equivalent to finding proper input space fuzzy clustering or, more precisely, to forming proper fuzzy hyperboxes in the input space. Initially, for each complement coded input vector $\mathbf{x}'$ [see (7)], the values of choice functions, T_j, are computed by

$$T_j(\mathbf{x}') = \frac{|\mathbf{x}' \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|}, j = 1, 2, \cdots, N \tag{20}$$

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

$$T_J = \max\{T_j : j = 1, \cdots, N\}. \tag{21}$$

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

$$\frac{|\mathbf{x}' \wedge \mathbf{w}_J|}{|\mathbf{x}'|} \geq \rho \tag{22}$$

where $\rho \in [0, 1]$ is a vigilance parameter. If the vigilance criterion is not met, we say mismatch reset occurs. In this case, the choice function value T_J is set to zero for the duration of the input presentation to prevent persistent selection of the same category during search (we call this action "disabling J"). A new index J is then chosen using (21). The search process continues until the chosen J satisfies (22). This search process is indicated by the feedback arrow marked with "vigilance test" in Fig. 7. If no such J is found, then a new input hyperbox is created by adding a set of n new input term nodes, one for each input linguistic variable, and setting up links between the newly added input term nodes and the input linguistic nodes. The complement weight vectors on these new layer-2 links are simply given as the current input vector, $\mathbf{x}'$. These newly added input term nodes and links define a new hyperbox, and thus a new category, in the input space. We denote this newly added hyperbox as J.

b) Output Fuzzy Clustering Process: The output fuzzy clustering process is exactly the same as the input fuzzy clustering process except that it is performed between layers 4 and 5 which are working in the up–down transmission mode. Of course, the training pattern used now is the desired output vector after complement coding, $\mathbf{r}' = (\bar{r}, \bar{r}^c) = (\bar{r}, 1 - \bar{r})$. We denote the chosen or newly added output hyperbox by K. This hyperbox is defined by the complement weight vector in layer 5, $\mathbf{w}_K = [(u_{1j}^{(5)}, 1 - v_{1j}^{(5)}), \cdots, (u_{ij}^{(5)}, 1 - v_{ij}^{(5)}), \cdots, (u_{mj}^{(5)}, 1 - v_{mj}^{(5)})]$.

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

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

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

Step 2) ELSE IF rule node J is not connected to output hyperbox K originally
THEN disable J and perform input fuzzy clustering process to find the next qualified J [i.e., the next rule node that satisfies (21) and (22)].
Go to Step 1).

Step 3) ELSE no structure change is necessary.

In the mapping process, hyperboxes J and K are resized according to the fast learning rule [24] by updating weights, $\mathbf{w}_J$ and $\mathbf{w}_K$, as follows:

$$\mathbf{w}_J^{(new)} = \mathbf{x}' \wedge \mathbf{w}_J^{(old)}, \qquad \mathbf{w}_K^{(new)} = \mathbf{r}' \wedge \mathbf{w}_K^{(old)}. \tag{23}$$

Note that once the consequent of a rule node has been decided in the mapping process, it will not be changed thereafter. We now use Fig. 5 to illustrate the structure-learning step as follows. For a given training datum, the input fuzzy clustering process and the output fuzzy clustering process find or form proper clusters (hyperboxes) in the input and output spaces. Assume that the input and output hyperbox pair found (or formed) are (J, K). The mapping process then tries to relate these two hyperboxes by setting up links between them. This is equivalent to finding a fuzzy logic rule that defines the association between an input hyperbox and an output hyperbox. If this association exists already (e.g., (J, K) = (IH1, OH1), (IH2, OH1), or (IH3, OH2) in Fig. 5), then no structural change is necessary. If input hyperbox J is

newly formed and thus not connected to any output hyperbox, it is connected to output hyperbox K directly. Otherwise, if input hyperbox J is associated with an output hyperbox different from K originally (e.g., $(J, K) = (\text{IH2}, \text{OH2})$), then a new input hyperbox close to J will be found or formed by performing the input fuzzy clustering process again. This search, called "match tracking" (see Fig. 7), continues until an input hyperbox, J', that can be associated with output hyperbox K is found [e.g., $(J', K) = (\text{IH3}, \text{OH2})$].

The vigilance parameter, ρ, is an important structure-learning parameter that determines learning cluster density. High (approaching one) ρ values tend to produce increasingly finer learning clusters, until at one, each training datum is assigned to its own cluster in the input (output) space. Low (approaching zero) ρ values tend to produce increasingly coarser learning clusters, until at zero, all training data are assigned to a single cluster in the input (output) space.

Clearly, a constantly high or low ρ value will result in formation of excessively high numbers of clusters on the one hand, or very low output accuracy (and thus, low network representation power) on the other hand. For these reasons, we chose an adaptive vigilance strategy in which the ρ parameter is initially set high to allow fast RFALCON structure growth, and then monotonically decreased to slow cluster formation and stabilize learning. Empirical studies have shown this approach to be efficient and stable in the learning speeds and numbers of clusters it produces.

2) The Parameter-Learning Step: After the network structure has been adjusted according to the current training pattern in the structure-learning step, it is then necessary to fine tune the network parameters using the same training pattern. This fine tuning process is necessary to assure the desired output accuracy of a network for control/prediction problems. Using the terminology of fuzzy logic: once our fuzzy controller has found its fuzzy logic rules, its membership functions must be tuned to make its output meet the desired output as closely as possible. Notice that the following parameter learning is performed on the whole network after structure learning, no matter whether the nodes (links) are newly added or are existent originally. The parameter-learning task can be stated as: Given the training input data $x_i(t), i = 1, \cdots, n$, the desired output value, and the network structure (specified by input and output hyperboxes and fuzzy logic rules), we need to adjust the network parameters to make the network output match the desired output values as closely as possible. Thus, the network works in a feedforward manner; that is, the nodes and links in layer 4 are in the down-up transmission mode. Basically, the backpropagation algorithm is used to find node output errors, which are then analyzed to guide parameter adjustment.

The goal of training the multistep critic network is to minimize the error function [see (19)]

$$E = \frac{1}{2}(r(t) + \xi p(t) - p(t-1))^2 \tag{24}$$

where $r(t)$ is the actual external reinforcement signal, and $p(t)$ is the predicted reinforcement signal. Gradient information is then easily derived

$$\left[\frac{\partial E}{\partial p}\right]_{t-1} = p(t-1) - r(t) - \xi p(t) = -\hat{r}(t) \tag{25}$$

where the subscript $t-1$ represents the time displacement. The time displacement in (25) and the remaining equations in this paper reflect the assumption that the reinforcement signal (which may be the "predicted" reinforcement signal) at time step t depends on the input state and chosen action at time step $t-1$.

We can derive the structure/parameter-learning algorithm for the multistep critic network using the following general parameter-learning rule:

$$w(t+1) = w(t) + \Delta w(t) = w(t) + \eta(-\frac{\partial E}{\partial w}), \tag{26}$$

$$-\frac{\partial E}{\partial w} = -\frac{\partial E}{\partial p}\frac{\partial p}{\partial w} \tag{27}$$

where w is the adjustable parameter in the critic network (i.e., u_{ij} or v_{ij}). The general parameter-learning rule then is

$$\Delta w(t) = \eta \hat{r}(t)\left[\frac{\partial p}{\partial w}\right]_{t-1}. \tag{28}$$

To show the parameter-learning rules, we derive the rules layer-by-layer using the hyperbox membership functions with corners u_{ij}'s and v_{ij}'s as the adjustable parameters for these computations. In the following derivation, we consider only one output linguistic variable for notational clarity. Hence, the adjustable parameters in layer 5 are denoted by $u_j^{(5)}, v_j^{(5)}$, and $m_j^{(5)} = (u_j^{(5)} + v_j^{(5)})/2$, for the jth term node.

Layer 5: Using (16), (26), and (27), the updating rule for the corners of the hyperbox membership function $v_j^{(5)}$ is

$$\frac{\partial E}{\partial v_j^{(5)}} = \frac{\partial E}{\partial a^{(5)}}\frac{\partial a^{(5)}}{\partial v_j^{(5)}} = -\hat{r}(t)\left[\frac{z_j^{(5)}}{2\sum z_j^{(5)}}\right]_{t-1}. \tag{29}$$

And the corner parameter is updated by

$$v_j^{(5)}(t+1) = v_j^{(5)}(t) + \eta \hat{r}(t)\left[\frac{z_j^{(5)}}{2\sum z_j^{(5)}}\right]_{t-1}. \tag{30}$$

Similarly, using (16), (26), and (27), the updating rule for the other corner parameter $u_j^{(5)}$ is

$$\frac{\partial E}{\partial u_j^{(5)}} = \frac{\partial E}{\partial a^{(5)}}\frac{\partial a^{(5)}}{\partial u_j^{(5)}} = -\hat{r}(t)\left[\frac{z_j^{(5)}}{2\sum z_j^{(5)}}\right]_{t-1}. \tag{31}$$

And the other corner parameter is updated by

$$u_j^{(5)}(t+1) = u_j^{(5)}(t) + \eta \hat{r}(t)\left[\frac{z_j^{(5)}}{2\sum z_j^{(5)}}\right]_{t-1}. \tag{32}$$

The error propagated to the preceding layer is

$$\delta^{(5)} = -\frac{\partial E}{\partial a^{(5)}} = -\frac{\partial E}{\partial p} = \hat{r}(t). \tag{33}$$

Layer 4: There is no parameter to be adjusted in this layer. Only the error signal ($\delta_i^{(4)}$) needs to be computed and propagated. According to (16), the error signal $\delta_i^{(4)}$ is derived by

$$\delta_i^{(4)} = -\frac{\partial E}{\partial a^{(4)}} = -\frac{\partial E}{\partial a^{(5)}}\frac{\partial a^{(5)}}{\partial a^{(4)}} \tag{34}$$

where

$$\frac{\partial E}{\partial a^{(4)}} = -\delta^{(5)}, \tag{35}$$

$$\frac{\partial a^{(5)}}{\partial a^{(4)}} = \frac{m_i^{(5)}\sum z_i^{(5)} - \sum m_i^{(5)} z_i^{(5)}}{(\sum z_i^{(5)})^2}. \tag{36}$$

Hence, the error signal is

$$\delta_i^{(4)} = \delta^{(5)}\left[\frac{m_i^{(5)}\sum z_i^{(5)} - \sum m_i^{(5)} z_i^{(5)}}{(\sum z_i^{(5)})^2}\right]_{t-1}. \tag{37}$$

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

Layer 3: As in layer 4, only the error signals need to be computed in this layer. According to (14), this error signal can be derived by

$$\delta_i^{(3)} = -\frac{\partial E}{\partial a^{(3)}} = -\frac{\partial E}{\partial a^{(4)}}\frac{\partial a^{(4)}}{\partial f^{(4)}}\frac{\partial f^{(4)}}{\partial a^{(3)}} \tag{38}$$

where

$$\frac{\partial E}{\partial a^{(4)}} = -\delta_i^{(4)}, \tag{39}$$

$$\frac{\partial a^{(4)}}{\partial f^{(4)}} = 1, \tag{40}$$

$$\frac{\partial f^{(4)}}{\partial a^{(3)}} = \frac{\partial f^{(4)}}{\partial z_i^{(4)}} = \frac{z_i^{(4)}}{z_{\max}} \tag{41}$$

where $z_{\max} = \max(\text{inputs of output terms node } j)$. The term $\frac{z_i^{(4)}}{z_{\max}}$ normalizes the error to be propagated for fired rules with the same consequent. Hence the error signal is

$$\delta_i^{(3)} = \delta_i^{(4)}\left[\frac{z_i^{(4)}}{z_{\max}}\right]_{t-1}. \tag{42}$$

If there are multiple outputs, then the error signal becomes $\delta_i^{(3)} = \sum_k \left[\frac{z_k^{(4)}}{z_{\max}}\right]_{t-1}\delta_k^{(4)}$, where the summation is performed over the consequents of a rule node; that is, the error of a rule node is the summation of the errors of its consequents.

Layer 2: Using (11), (26), and (27), the updating rule of $v_{ij}^{(2)}$ is derived as in the following:

$$-\frac{\partial E}{\partial v_{ij}^{(2)}} = -\frac{\partial E}{\partial a^{(3)}}\frac{\partial a^{(3)}}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial v_{ij}^{(2)}} \tag{43}$$

where

$$\frac{\partial a^{(3)}}{\partial a^{(2)}} = 1, \tag{44}$$

$$\frac{\partial a^{(2)}}{\partial v_{ij}^{(2)}} = \begin{cases} \frac{\gamma}{n} & \text{if } 0 \le (\bar{x}_i - v_{ij}^{(2)})\gamma \le 1. \\ 0 & \text{otherwise.} \end{cases} \tag{45}$$

So the adaptive rule of $v_{ij}^{(2)}$ is

$$v_{ij}^{(2)}(t+1) = v_{ij}^{(2)}(t) + \eta\delta_i^{(3)}\left[\frac{\partial a^{(2)}}{\partial v_{ij}^{(2)}}\right]_{t-1}. \tag{46}$$

Similarly, using (11), (26), and (27), the updating rule of $u_{ij}^{(2)}$ is derived as

$$-\frac{\partial E}{\partial u_{ij}^{(2)}} = -\frac{\partial E}{\partial a^{(3)}}\frac{\partial a^{(3)}}{\partial a^{(2)}}\frac{\partial a^{(2)}}{\partial u_{ij}^{(2)}} \tag{47}$$

where

$$\frac{\partial a^{(3)}}{\partial a^{(2)}} = 1, \tag{48}$$

$$\frac{\partial a^{(2)}}{\partial u_{ij}^{(2)}} = \begin{cases} -\frac{\gamma}{n} & \text{if } 0 \le (u_{ij}^{(2)} - \bar{x}_i)\gamma \le 1 \\ 0 & \text{otherwise.} \end{cases} \tag{49}$$

Hence, the adaptive rule of u_{ij} becomes

$$u_{ij}^{(2)}(t+1) = u_{ij}^{(2)}(t) + \eta\delta_i^{(3)}\left[\frac{\partial a^{(2)}}{\partial u_{ij}^{(2)}}\right]_{t-1}. \tag{50}$$

The proposed reinforcement learning algorithm provides a novel on-line scheme to combine the structure and the parameter learning such that they can be performed simultaneously.

B. The Action Network

In this section, we develop the reinforcement learning algorithm for the action network. The goal of this reinforcement structure/parameter-learning algorithm is to adjust the parameters (e.g., w's) of the action network, to change the connection types, or even to add new nodes, if necessary, such that the reinforcement signal is maximum; that is

$$\Delta w \propto \frac{\partial r}{\partial w}. \tag{51}$$

To determine $\frac{\partial r}{\partial w}$, 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, and 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 developed in the last section for the critic network 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 randomness in how output actions are chosen by the action network so the range of possible outputs 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.

In our learning algorithm, the gradient information, $\frac{\partial r}{\partial y}$, is also estimated by the stochastic exploration method [32]. In particular, the intuitive idea behind multiparameter distributions suggested by Williams [17] 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. The stochastic exploration, $\sigma(t)$, is some nonnegative, monotonically decreasing functions of the predicted reinforcement signal, e.g., $1/(1+\exp(2p(t)))$. The $\sigma(t)$ can be interpreted as the extent to which the output node searches for a better action. Since $p(t)$ is the predicted reward signal, if $p(t)$ is small, the exploratory range, $\sigma(t)$, will be large. 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 predicted reinforcement signal is large. This can provide a higher probability of choosing 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 predicted reinforcement signal is small such that the actual action has a higher probability of being quite different from the mean action, $y(t)$. Thus, if an expected action has a smaller predicted 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 [35], a Gaussian search was also used in developing a stochastic reinforcement learning algorithm, where the variance was chosen as a decreasing function of the predicted reinforcement signal. This idea was later used in [33] for the reinforcement learning of a ball-and-beam system, which will be mentioned in Section IV.

In the above two-parameter distribution approach, a predicted reinforcement signal is necessary to determine the search range, $\sigma(t)$. This predicted reinforcement signal can be obtained from the critic network. Once the variance has been decided, the actual output of the stochastic node can be set as

$$\hat{y}(t) = N(y(t), \sigma(t)). \tag{52}$$

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

$$Prob(\hat{y}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\{\frac{-(\hat{y}-y)^2}{2\sigma^2}\}. \tag{53}$$

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.

From the above discussion, the gradient information is estimated as

$$\frac{\partial r}{\partial y} \approx \hat{r}(t)\left[\frac{\hat{y}(t-1)-y(t-1)}{\sigma(t-1)}\right] \equiv \hat{r}(t)\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1} \tag{54}$$

where the subscript, $t-1$, represents time displacement. We can observe that if $\hat{r}(t) > 0$, 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 $\hat{r}(t) < 0$, 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 $\hat{y}(t-1)$.

Basically, the training of a action network is not a supervised learning problem. There are no correct "target" output values for each input pattern. In action network structure learning, however, we need the desired output values to determine proper output fuzzy partitions as well as membership functions and to find fuzzy logic rules. The desired output values can be estimated as

$$y_d(t) \approx y(t) + \kappa\frac{\partial r}{\partial y} \tag{55}$$

where κ is a real number in the range $[0,1]$ and $\frac{\partial r}{\partial y}$ can be replaced by $\hat{r}(t)\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}$ in (54). According to the input state values and the estimated desired output values, the structure-learning step described in Section III-A1) can be performed on the action network directly.

The goal of action network parameter learning 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 above. With the predicted reinforcement signal, $p(t)$, and the internal reinforcement signal, $\hat{r}(t)$, provided from the critic network, the action network can perform stochastic exploration and learning. The prediction signal $p(t)$ is used to determine the variance of the normal distribution function during stochastic exploration. Then the actual output, $\hat{y}(t)$, can be determined according to (52) and the gradient information can be obtained by (54).With this gradient information, action network parameter learning can be performed in exactly the same way as that of the critic network described in Section III-A2). The exact action network parameter-learning equations are the same as (29)–(50) except that $\hat{r}(t)$ is replaced by the new error term $\hat{r}(t)\left[\frac{\hat{y}-y}{\sigma}\right]_{t-1}$. As a summary, the flowchart of the reinforcement learning algorithm for the action network is shown in Fig. 7. Basically, we use the stochastic exploration method to find the output error of the action network. The error is then used for training the action network by the above two-step (structure-learning step and parameter-learning step) learning algorithm.

IV. Illustrative Examples

A general purpose simulator for the RFALCON model with the multistep critic network has been written in the C Programming and runs on an 80 486-DX-based personal computer. Using this simulator, two typical examples are presented in this section to show the fundamental applications of the proposed model: the cart-pole balancing system [12] and the ball and beam system [33], [37].

Example 1—Control of the Cart-Pole System: The cart-pole balancing problem involves of learning how to balance an upright pole as shown in Fig. 8. The bottom of the pole is hinged to a cart that travels along a finite-length track to its right or its left. Both the cart and pole can move only in the vertical plane; that is, each has only one degree of freedom. There are four input state variables in this system: θ, angle of the pole from an upright position (in degrees); $\dot{\theta}$, angular

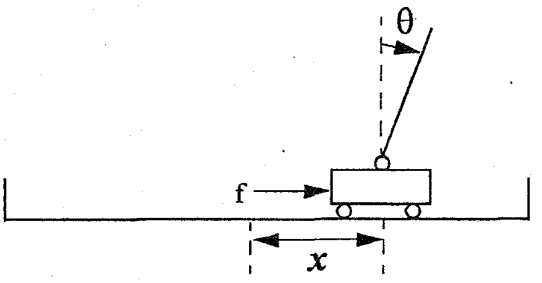


Fig. 8. The cart-pole balancing system.

velocity of the pole (in degrees/second); x, horizontal position of the cart's center (in meters); and $\dot{x}$, velocity of the cart (in meters/second). The only control action is f, which is the amount of force (in Newtons) applied to the cart to move it left or right. The system fails and receives a penalty signal of -1 when the pole falls past a certain angle ($\pm 12°$ was used) 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 RFALCON to determine the sequence of forces and magnitudes to apply to the cart to balance the pole for as long as possible without failure.

The model and the corresponding parameters of the cart-pole balancing system for our computer simulation are adopted from [12,19] with the consideration of friction effects. The equations of motion that we used are given in (56)–(59), shown at the bottom of the page, where

g	$= -9.8m/s^2$	Acceleration due to the gravity
m	$= 1kg$	Mass of the cart
m_p	$= 0.1kg$	Mass of the pole
l	$= 0.5m$	Half-pole length
μ_c	$= 0.0005$	Friction coefficient of cart on track
μ_p	$= 0.000002$	Friction coefficient of pole on cart
Δ	$= 0.02$	Sampling interval.

(60)

The constraints on the variables are $-12° \leq \theta \leq 12°$, $-2.4m \leq x \leq 2.4m$, and $-10N \leq f \leq 10N$. 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 only when a failure occurs; that is, either $|\theta| > 12°$ or $|x| > 2.4m$. Since no exact teaching information is available, this is a typical reinforcement learning problem and the feedback failure signal serves as the reinforcement signal. Since a reinforcement signal may only be available after a long sequence of time steps in this failure avoidance task, a multistep critic network is required for the RFALCON. 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 RFALCON can balance the pole for as long as possible for an infinite number of trials, not just for one particular trial, where a "trial" is defined as the time steps from an initial state to a failure. Hence, the cart-pole balancing problem is an infinitely discounted prediction problem, and (17) should be used for temporal difference prediction. The reinforcement signal is defined as

$$r(t) = \begin{cases} -1, & \text{if } |\theta(t)| > 12° \text{ or } |x(t)| > 2.4m \\ 0, & \text{otherwise} \end{cases} \quad (61)$$

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

In the simulations, the learning system was tested for five runs. Each run consisted of a sequence of trials; each trial began with the same initial condition and ended with a failure signal indicating that either $|\theta| > 12°$ or $|x| > 2.4m$. A run consisted of at most 60 trials, unless the duration of each run exceeded 50 000 time steps. In the latter case, we considered the run "successful." The following learning parameters were used for each trial. The learning rate $\eta = 0.001$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.5, \rho_{\text{output}} = 0.8$ were used for the action network. The learning rate $\eta = 0.002$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.4, \rho_{\text{output}} = 0.7$ were used for the critic network, where ρ_{input} was the vigilance parameter used in the input fuzzy clustering process and ρ_{output} was the vigilance parameter used in the output fuzzy clustering process.

In our computer simulations, a total of five runs were performed. Each run started at a different initial state. The simulation results in Fig. 9 shows that the RFALCON learned to balance the pole at the 15th trial on average. Fig. 10 shows the pole position (angular deviation of the pole) when the cart-pole system was controlled by a well-trained RFALCON starting at the initial state: $\theta(0) = 5.8, \dot{\theta}(0) = 0.058, x(0) = 0.1, \dot{x}(0) = 0.01$. The average angular deviation was 0.5°. On average, there were ten fuzzy logic rules generated in the action network and six fuzzy logic rules generated in the critic

$$\theta(t+1) = \theta(t) + \Delta\dot{\theta}(t), \quad (56)$$

$$\dot{\theta}(t+1) = \dot{\theta}(t) + \Delta\frac{(m+m_p)g\sin\theta(t) - \cos\theta(t)[f(t) + m_p l\dot{\theta}(t)^2\sin\theta(t) - \mu_c sgn(\dot{x}(t))] - \frac{\mu_p(m+m_p)\dot{\theta}(t)}{m_p l}}{(4/3)(m+m_p)l - m_p l\cos^2\theta(t)}, \quad (57)$$

$$x(t+1) = x(t) + \Delta\dot{x}(t), \quad (58)$$

$$\dot{x}(t+1) = \dot{x}(t) + \Delta\frac{f(t) + m_p l[\dot{\theta}(t)^2\sin\theta(t) - \ddot{\theta}(t)\cos\theta(t)] - \mu_c sgn(\dot{x}(t))}{(m+m_p)} \quad (59)$$

TABLE I
PERFORMANCE COMPARISON OF VARIOUS REINFORCEMENT SYSTEMS

	Ours	Barto's original system	Barto's system with continuous output	Neural Networks	Fuzzy Neural Network		
					Fuzzy rules must be given in advance		
				C.W. Anderson	Lee and Berenji	Berenji and Khedkar	Lin and Lee
Trials	15	35	40	8000	10	300	10
Rule number	10	162	162	–	189	13	35
Angular deviation	0.5°	4°	2.5°	–	2°	1°	1°

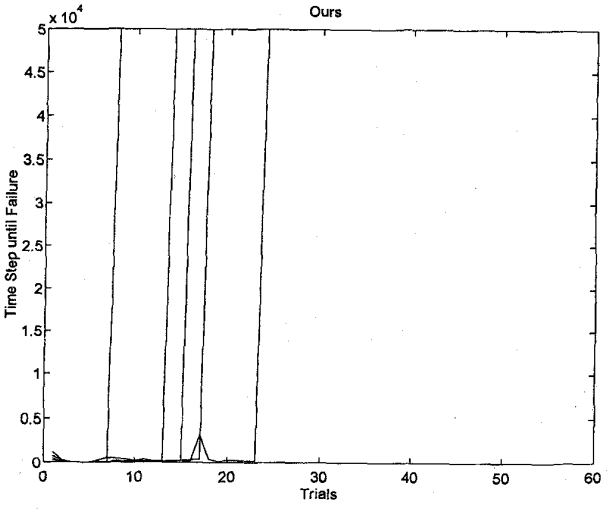


Fig. 9. Performance of the RFALCON on the cart-pole balancing problem.

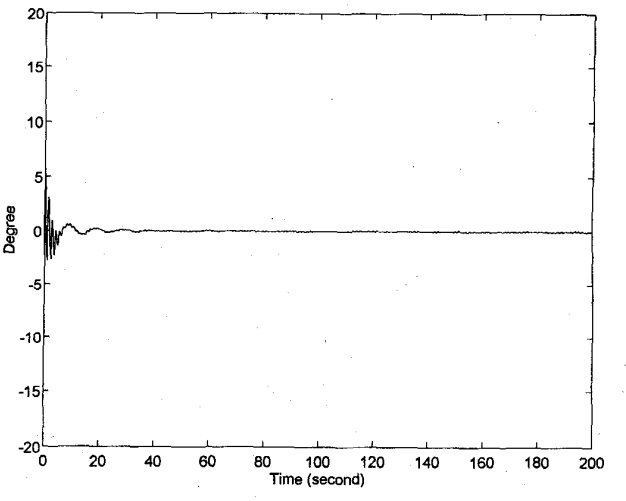


Fig. 10. Angular deviation of the pole allowed by a trained RFALCON.

network after learning. These results are shown in the first column of Table I.

We now compare the performance of our system with that of other existing reinforcement learning systems. The performance indexes considered include number of trials, numbers of fuzzy rules, and angular deviation of the pole. The detailed comparison is tabulated in Table I. In this table, the performance of various systems is averaged over runs. First, we compare the performance of the RFALCON with that of the Barto's original system [12]. Two neuron-like adaptive elements are integrated in this system. They are the associative search element (ASE) used as a controller, and the adaptive critic element (ACE) used as a predictor. Temporal difference techniques and single-parameter stochastic exploration are used in this system. A bang-bang control scheme is used in the ASE, where a threshold function is used for force output, thus, the control output can have only two values: $\pm 10N$. They divided the four-dimensional state space of the cart-pole system into disjoint regions (or boxes) by quantizing the four state variables. They distinguished three grades of cart position, six of pole angle, three of cart velocity, and three of pole angular velocity. This yielded $3 \times 6 \times 3 \times 3 = 162$ regions corresponding to all of the combinations of intervals. Each box was imagined to contain a local demon whose job is to choose a control action whenever the system state enters its box. Hence, there were the equivalent of 162 (nonfuzzy) control rules, one for each box. Fig. 11 illustrates the simulation results for Barto's original system [12]. A total of five runs were performed in the simulations. Each run started at the same initial states as those set for the RFALCON, and the results show that this system learned to balance the pole at the 35th trial on average. Additional observations are made on the state trajectory of the angular deviation of the pole with respect to the vertical plane. Fig. 12 shows the angular deviation of the cart-pole system starting at the initial state: $\theta(0) = 5.8, \dot{\theta}(0) = 0.058, x(0) = 0.1, \dot{x}(0) = 0.01$. The average angular deviation is about 4 °. The results show that our controller is able to keep the pole angle within a smaller region than Barto's original system. These results are shown in the second column of Table I. The simulation results also show that our system has better learning performance than Barto's original system.

In another set of simulations, we changed the threshold function in the ASE of Barto's original system to a sigmoid function such that the force output became continuous in the range $[-10N, +10N]$. In this system, the input space was also partitioned into 162 regions. Fig. 13 illustrates the simulation results based on Barto's system with continuous output, and show that this system learned to balance the pole at the 40th trial on average. This is a little longer than Barto's original system using discrete output. The angular deviation of the pole about the center point is shown in Fig. 14. The average angular deviation was about 2.5 °. Hence, although Barto's system with continuous output needs more trials than the original system, the angular deviation produced by it is smaller. These results are shown in the third column of Table I. Our RFALCON's learning performance is still better than that of Barto's system even with continuous output.

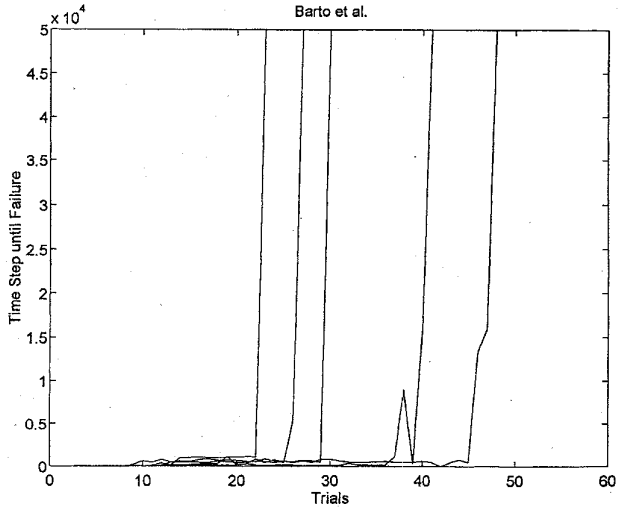


Fig. 11. Performance of Barto's original system on the cart-pole balancing problem.

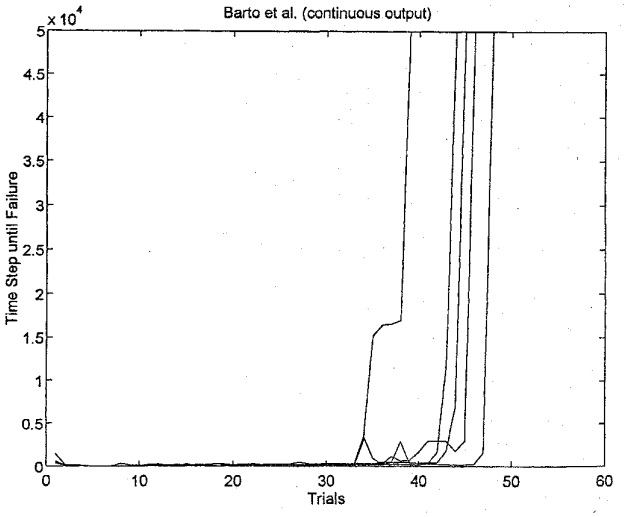


Fig. 13. Performance of Barto's system with continuous output on the cart-pole balancing problem.

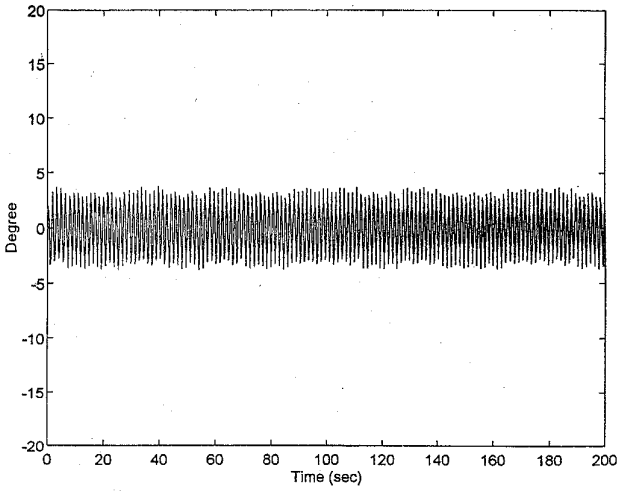


Fig. 12. Angular deviation of the pole allowed by a trained Barto's original system.

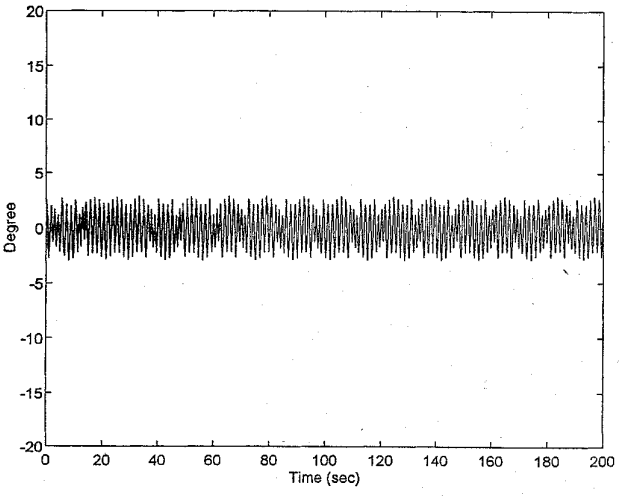


Fig. 14. Angular deviation of the pole allowed by a trained Barto's system with continuous output.

Anderson [19] extended the work of Barto *et al.* by replacing the ASE and ACE with two two-layer feedforward networks. Hence the force output in his system is also continuous. A form of error backpropagation was derived to train both networks to learn to balance the pole given the actual state values of the cart-pole system as input. As shown in Table I, this approach required about 8000 trials to control the cart-pole system. It took more trials than the other approaches listed in Table I due to the slow convergence of the pure backpropagation algorithm. In [19], we can find no further information about the number of hidden nodes used in the system and the average angular deviation produced by it.

Lee and Berenji [28] modified Barto's original system by introducing fuzziness into it. They first partitioned the input space into boxes and then defined membership function for each box. These membership functions overlap one another to allow generalization to occur beyond the confines of a given box. They partitioned the input space into 189 boxes ($3 \times 7 \times 3 \times 3$), which yielded the equivalent of 189 fuzzy logic rules, one for each overlapping box in the input space. As shown in Table I, the average number of trials with this approach is about 10 and the average angular deviation is 2°.

Berenji and Khedkar [18] proposed a model, called generalized approximate reasoning-based intelligent control (GARIC) architecture, for learning and tuning a fuzzy controller based on reinforcement signals from a dynamical system. Their architecture extended Anderson's method [19] by including *a priori* control knowledge of expert operators in the form of fuzzy control rules. In the GARIC, a three-layer feedforward neural network, called the action evaluation network (AEN), is used to predict external reinforcement signals and produce internal reinforcement signals. The role of the AEN parallels that of the ACE in Barto's system and the critic network in our system. A five-layer feedforward neural network, called the

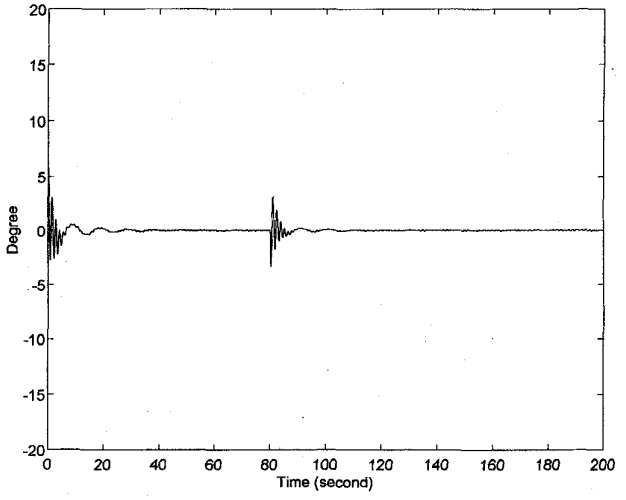


Fig. 15. Angular deviation of the pole allowed by a trained RFALCON after a disturbance is given.

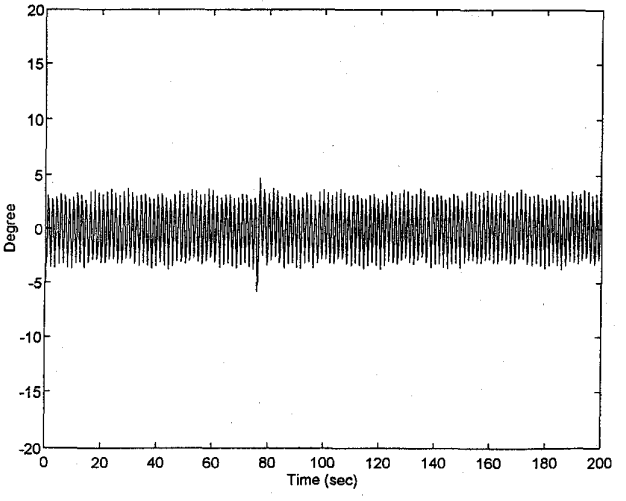


Fig. 16. Angular deviation of the pole allowed by a trained Barto's original system after a disturbance is given.

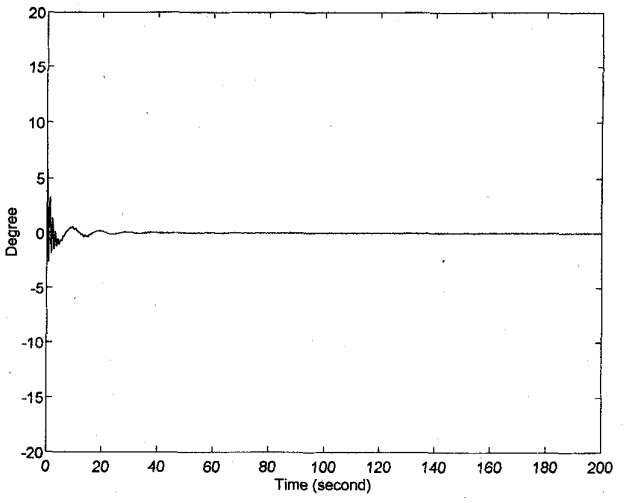


Fig. 17. Angular deviation of the pole allowed by a trained RFALCON when the half-length of the pole is reduced to 0.25 m from 0.5 m.

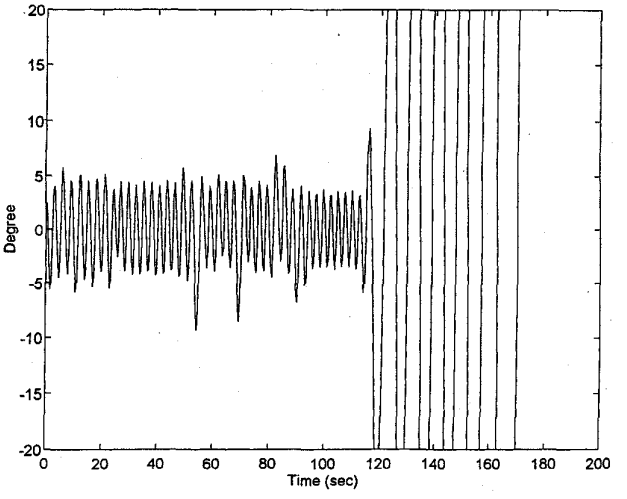


Fig. 18. Angular deviation of the pole allowed by a trained Barto's original system when the half-length of the pole is reduced to 0.25 m from 0.5 m.

action selection network (ASN), is used as a fuzzy controller. The role of the ASN parallels that of the ASE in Barto's system and the action network in our system. In the ASN, a total of 13 fuzzy rules are used to control the cart-pole system as indicated in Table I. This set of correct fuzzy logic rules, however, must be given in advance by experts before initiating training of the GARIC [18]. As shown in Table I, this approach required about 300 trials to control the cart-pole system. The average angular deviation of the pole about the center point produced by the GARIC was about 1 °.

In [20], Lin and Lee proposed an RFALCON, two connectionist fuzzy systems are used for the controller and the predictor, respectively. This system has the ability to find proper network structures (fuzzy rules) and parameters (membership functions) simultaneously and dynamically. They still partitioned the input and output spaces into grids, however. They needed 35 fuzzy logic rules to control the cart-pole system. This number is larger than that given by experts (as in the GARIC) or that learned by the proposed RFALCON. As shown in Table I, this system required about 10 trials to control the cart-pole system. The average angular deviation of the pole about the center point produced by the RNN-FLCS is about 1 °.

Similar to the simulation in [18], the adaptation capability of the proposed RFALCON and Barto's original system were tested and compared. To demonstrate the disturbance rejection capability of the trained system, we gave a disturbance $f = 10N$ to the cart at the 80th second. The trajectories of angular deviations in Figs. 15 and 16 indicate that both the RFALCON and Barto's original system brought the pole back to the center position quickly after the disturbance was given. During the process, neither system required any further trials for relearning. We also changed the parameters of the cart-pole system to test robustness. We first reduced the pole length, l, from $0.5m$ to $0.25m$. Figs. 17 and 18 show, respectively, the results produced by the RFALCON, and those produced by Barto's original system. We found that the latter cannot keep angular deviation within the range $[-12°, +12°]$ without

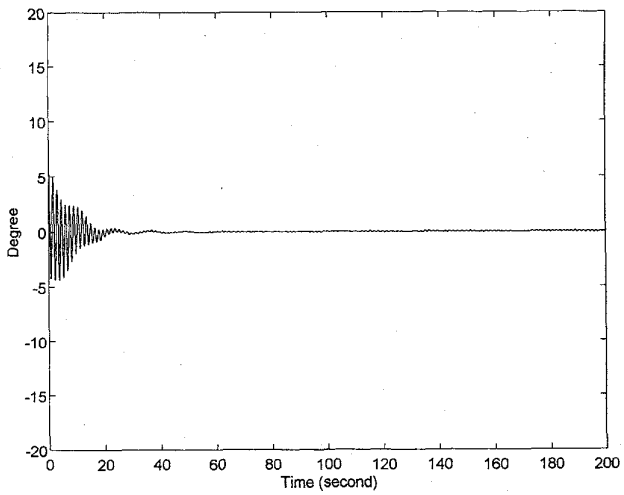


Fig. 19. Angular deviation of the pole allowed by a trained RFALCON when cart mass is doubled to 2 kg from 1 kg.

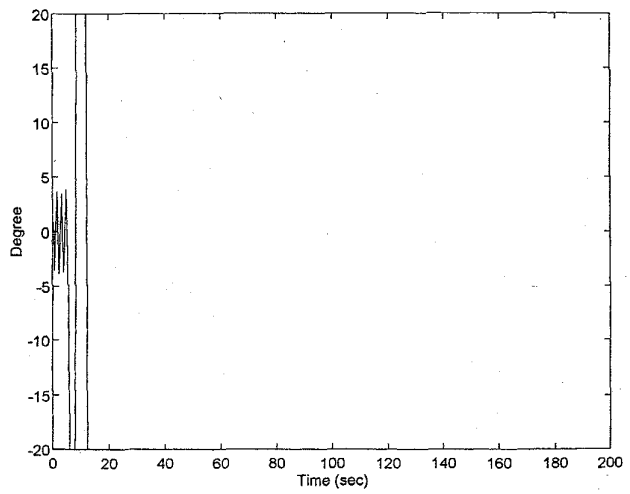


Fig. 20. Angular deviation of the pole allowed by a trained Barto's original system when cart mass is doubled to 2 kg from 1 kg.

relearning, but the proposed RFALCON can still balance the pole without any relearning. In another test, we doubled the mass of the cart, m, to 2 kg from 1 kg. Figs. 19 and 20 show, respectively, the results produced by the RFALCON, and those produced by Barto's original system. Again, the results show that Barto's original system cannot keep angular deviation within the range $[-12°, +12°]$ without relearning, but the proposed RFALCON can still balance the pole without any relearning. These robustness tests show that no further trials are required for relearning by a trained RFALCON when the controlled system parameters are changed in the ways mentioned. Further trials are required, however, for relearning in Barto's original system. The results show the good control and disturbance rejection capabilities of the trained RFALCON in the cart-pole balancing system.

Example 2—Control of the Ball and Beam System: The ball and beam system is shown in Fig. 21. The beam is made to rotate about a horizontal axis by applying a torque at the center of rotation and the ball is free to roll along the beam.

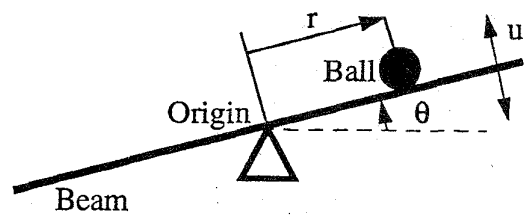


Fig. 21. The ball and beam system.

We require that the ball remains in contact with the beam. There are four input state variables in this system: r, position of the ball's center from origin (in meters); $\dot{r}$, velocity of the ball (in meters/second); θ, angle of the beam from the horizontal (in degrees); and $\dot{\theta}$, angular velocity of the beam (in degrees/second). The only control action, u, is the angular acceleration. The system fails and receives a penalty signal of -1 when the beam deviates beyond a certain angle ($\pm 14°$ is used here) or the ball reaches the end of the beam (the distance is four meters from the origin to both ends of the beam). The goal of this control problem is to train the RFALCON to determine the proper sequence of forces and magnitudes to apply to the beam to balance the ball for as long as possible without failure.

The ball and beam system can be described by a state-space form as

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} x_2 \\ B(x_1 x_4^2 - G \sin x_3) \\ x_4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u, \tag{62}$$

$$y = x_1 \tag{63}$$

where $(x_1, x_2, x_3, x_4) \equiv (r, \dot{r}, \theta, \dot{\theta})$ is the state of the system and $y = x_1 \equiv r$ is the output of the system. The control signal u is the angular acceleration $(\ddot{\theta})$ and the parameters $B = 0.7143$ and $G = 9.81$ are chosen in this system. These parameters are also used in [33] and [34]. The purpose of control is to determine $u(t)$ such that the system output y will converge to zero from different initial conditions. The constraints on the variables are $-4m \leq x_1 \leq 4m$, $-14° \leq x_3 \leq 14°$, and $-70N \leq u \leq 70N$.

In designing the controller, the state-space equations of the ball and beam 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 only when a failure occurs; that is, either $|x_1| > 4m$ or $|x_3| > 14°$. Since no exact teaching information is available, this is a typical reinforcement learning problem and the feedback failure signal serves as the reinforcement signal. Since a reinforcement signal may only be available after a long sequence of time steps in this failure avoidance task, a multistep critic network is required for the RFALCON. 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 RFALCON can balance the ball for as long as possible for an infinite number of trials, not just for one particular trial, where a "trial" is defined as the time steps from an initial state to a failure. Thus, like the cart-pole balancing problem, the ball and beam balancing problem is a typical infinitely

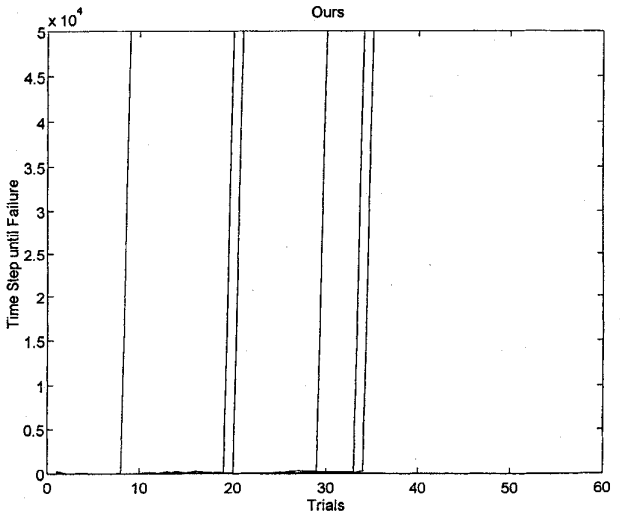


Fig. 22. Performance of the RFALCON on the ball and beam problem. The constraints on the variables are $-4m \leq x_1 \leq 4m$, $-14^\circ \leq x_3 \leq 14^\circ$.

discounted prediction problem, and (17) should be used for temporal difference prediction. The reinforcement signal is defined as

$$r(t) = \begin{cases} -1, & \text{if } |x_1(t)| > 4m \text{ or } |x_3(t)| > 14^\circ \\ 0, & \text{otherwise} \end{cases} \quad (64)$$

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

In our computer simulations, the learning rate $\eta = 0.001$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.7, \rho_{\text{output}} = 0.7$ for the action network and the learning rate $\eta = 0.002$, sensitivity parameter $\gamma = 4$, and initial vigilance parameter $\rho_{\text{input}} = 0.7, \rho_{\text{output}} = 0.7$ for the critic network are chosen. A total of six runs were performed in the simulation. Each run started at a different initial state, and consisted of a sequence of trials, each beginning with the same initial conditions and ending with a failure signal indicating that either $|x_1| > 4m$ or $|x_3| > 14^\circ$. A run consisting of at most 60 trials, was considered successful and terminated after 50 000 time steps. The simulation results in Fig. 22 show that the RFALCON learned to balance the ball at the 20th trial on average. Fig. 23 shows the ball position (deviation of the ball from the center point) when the ball and beam system was controlled by a well-trained RFALCON starting from the initial state: $[-1.2, -0.01, 0.58, 0.58]$. On average, there were fourteen fuzzy logic rules generated in the action network and nine fuzzy logic rules generated in the critic network after learning. In another six runs, we set the constraints on the variables as $-2m \leq x_1 \leq 2m$, $-12^\circ \leq x_3 \leq 12^\circ$, and kept the other two constraints unchanged. The simulation results shown in Fig. 24 indicate that the RFALCON could still balance the ball at the center position of the plane within 45 trials under the new constraints.

For comparison, we also used Barto's original system [12] on the ball-and-beam balancing problem. We divided the four-dimensional state space of the ball and beam system into disjoint regions (or boxes) by quantizing the four state

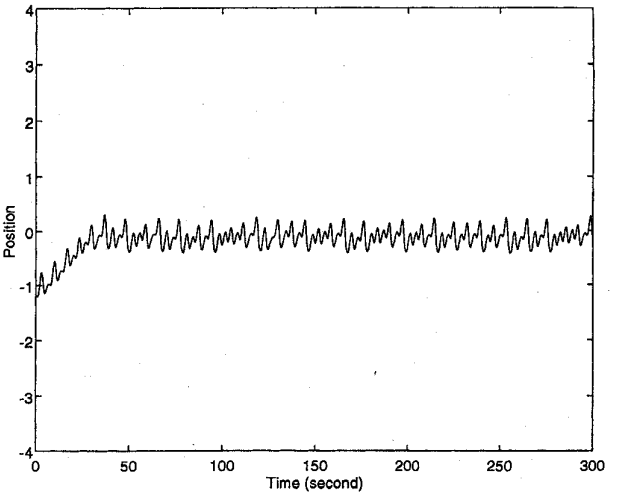


Fig. 23. Position deviation of the ball produced by a trained RFALCON.

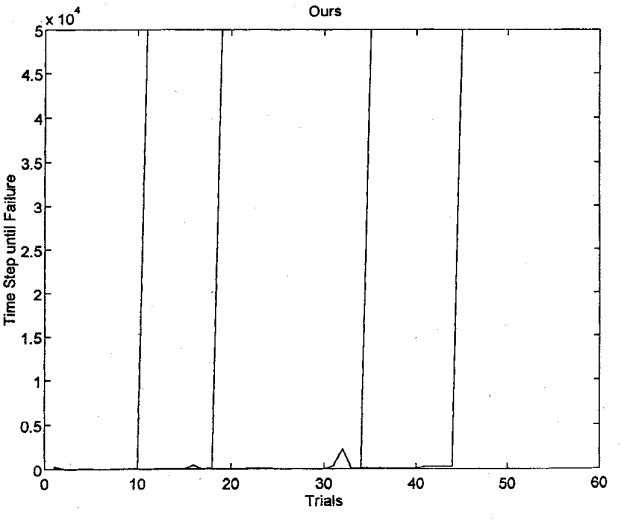


Fig. 24. Performance of the RFALCON on the ball and beam problem. The constraints on the variables are $-2m \leq x_1 \leq 2m$, $-12^\circ \leq x_3 \leq 12^\circ$.

variables. After several simulations, we finally distinguished three grades of ball position, six of beam angle, three of ball velocity, and three of beam angular velocity, yielding $3 \times 6 \times 3 \times 3 = 162$ regions corresponding to all of the combinations of the intervals. Each box is imagined to contain a local demon whose job is to choose a control action whenever the system state enters its box. Fig. 25 illustrates the simulation results of using Barto's original system. We see that each run failed within 8000 time steps. It was found that proper definition and alignment of the partitioned regions are critical to Barto's original system. Benbrahim *et al.* [33] also applied Barto's original system to the ball and beam balancing problem. They distinguished five grades of ball position, three of beam angle, six of ball velocity, and two of beam angular velocity, yielding $5 \times 3 \times 6 \times 2 = 180$ boxes. The time step until failure is topped at 300, where 300 steps represent about 15 s of real time. Gullapalli *et al.* [34] also proposed a structure based on Barto's ASE-ACE configuration for reinforcement

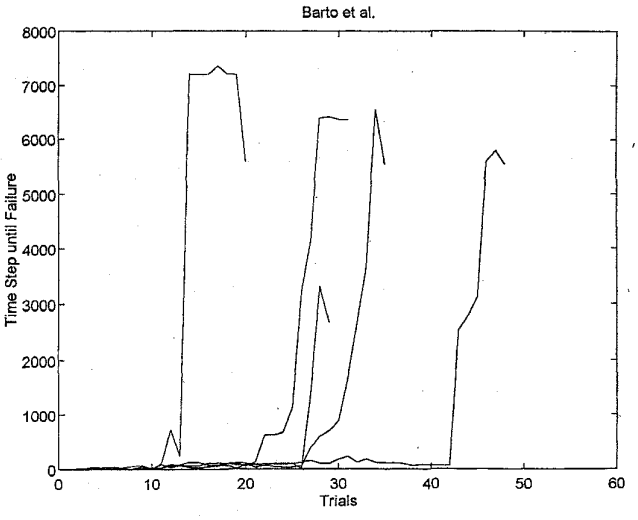


Fig. 25. Performance of Barto's original system on the ball and beam problem.

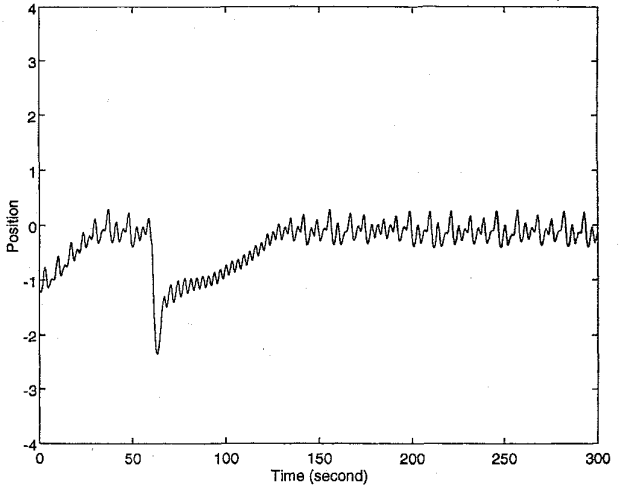


Fig. 26. Position deviation of the ball allowed by a trained RFALCON after the disturbance is given.

learning. The ASE used was a multilayer feedforward network with a stochastic real-valued (SRV) output unit that outputs the control action as the motor voltage. The hidden layer is a set of boxes representing the quantized state. There are 180 hidden units. The hidden layer becomes the set of inputs used by the SRV unit. This controller successfully learned to balance the ball and improved over the two-action controller (i.e., Barto's original system). Their learning curve in [34] showed that the ball was balanced with no further failure after 700 failures.

In the published literature, we could not find a paper that used reinforcement learning to train a fuzzy controller to solve the ball and beam balancing problem. Wang and Mendel [37], however, proposed an adaptive fuzzy controller that can learn to control the ball and beam system by supervised learning. In their approach, 20 fuzzy logic rules were used to balance the ball, where our system, the RFALCON, used 14 fuzzy logic rules to balance the ball by reinforcement learning.

We then tested the adaptation capability of the proposed RFALCON on the ball and beam system. To demonstrate the

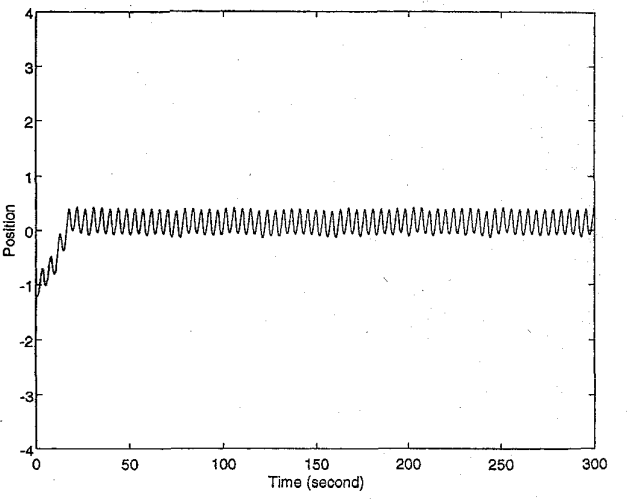


Fig. 27. Position deviation of the ball allowed by a trained RFALCON when ball mass is reduced to 0.025 kg from 0.05 kg.

disturbance rejection capability of the trained system, we gave a disturbance $u = 60N$ to the ball and beam system at the 60th second. The results shown in Fig. 26 indicate that the RFALCON brought the ball back to the origin quickly after the disturbance was given. No further training was required during the process. This figure demonstrates the good disturbance rejection capability of the trained RFALCON. Paralleling the robustness tests performed in Example 1, we also changed the parameters of the ball and beam system. Fig. 27 shows the simulation results obtained when the ball's mass was reduced to $0.025kg$ from $0.05kg$, i.e., $B = 0.556$, and Fig. 28 shows the simulation results obtained when the ball's mass was doubled to $0.1kg$ from $0.05kg$, i.e., $B = 0.833$. No further trials were required for relearning in these tests. The results show the good control and disturbance rejection capabilities of the trained RFALCON in the ball and beam system.

V. Discussion

In this section, we summarize the features of the proposed RFALCON. It keeps the ability of Barto's original system [12] to learn at each time step within a trial without waiting to know the actual outcome, by using a multistep critic network. In addition, distributed representation is used to represent the input vectors in the RFALCON. 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 partitioning 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 used in the BOXES system [30] and in Barto's system [12], which divides the input space into disjoint regions (boxes) and allows generalization only within the confines of a given box. The dynamic fuzzification process in the RFALCON also avoids the necessity of partitioning the input space into disjoint [12] or overlapping [28] small regions in advance.

The second important feature of the proposed RFALCON is its dynamic structure/parameter-learning ability that can find proper fuzzy logic rules. This is in contrast to the approach in [18], which needs *a priori* control knowledge from expert operators in term of fuzzy control rules. The third feature of the proposed RFALCON is its capacity for stochastic search with multiparameter distribution functions. This gives the action network a higher probability of finding a better action via the prediction signal from the critic network instead of limiting it to perform random searches around the expected action with single-parameter distribution functions. The fourth feature of the proposed RFALCON is the ease with which expert knowledge can be incorporated into its the action network and the critic network greatly shortening learning time. The fifth important feature of the proposed RFALCON is that it flexibly partitions the input–output spaces according to the distribution of environment states and reinforcement signals. This avoids the combinatorial growth problem encountered by partitioned grids in some complex systems.

Using the Stone–Weierstrass theorem [38], we have shown that the structure of the FALCON is a universal approximator (the detailed proof will be presented in a future paper). Although we have not provided a convergence theorem demonstrating the stability of the proposed on-line (supervised or reinforcement) structure/parameter-learning algorithm yet, many empirical studies have shown the proposed learning scheme to be stable. In addition to the simulations done in this paper, the proposed on-line supervised structure/parameter-learning algorithm has been used to identify dynamic systems, predict chaotic time-series, and control of the truck backer-upper. Some of these results were presented in [23], [2]. Furthermore, this algorithm has been stable when used for practical position control of a crane in our lab. These empirical studies and the following analysis lead us to be optimistic about the stability of our learning scheme. The basic concept behind our learning algorithm is that structure learning determines the skeleton of the network, while parameter learning fine tunes the network parameters to achieve the desired output control accuracy. Structure learning usually causes large weight changes if there are changes in a learning time step [see (23)], whereas parameter learning often causes smaller weight changes in a learning time step due to its gradient descent nature and small learning constant. Hence, although the two learning steps may control the values of the same weights, they produce cooperative effects on weight changes. The major effect of structure learning on weight changes is definition of new hyperboxes (and thus new weights), or resizing of hyperboxes by embedding training data directly into the weights [see (23)], whereas the major effect of the parameter learning is to fine tune the weights around the values determined in the structure-learning step. Furthermore, the use of adaptive (monotonically decreasing) vigilance values is also important to the stability of the proposed learning scheme. As mentioned at the end of Section III-A1), decreasing vigilance values will cause fast structure growth in the early stages of learning, and supervised weight tuning of a stable network structure in the later stages of learning. Hence, the two learning steps interact, each playing the major role at different learning stages. This can also greatly speed up parameter learning since structure learning has built up the regularity of the training data in the early stages of learning, and parameter learning just fine tunes the parameters based on the *a priori* knowledge obtained in the structure-learning step. This becomes clear when we perform the two learning steps separately, as in cases where we have a whole set of training data and learning can be performed off-line. That is, we first perform structure learning on all training data, and then perform parameter learning using all training data. In such cases, there should be no instability problem, since the convergence of the ART-like learning in the structure-learning step has been demonstrated conclusively in [24], and the gradient descent procedure in the parameter-learning step can be expected to eventually converge to values that minimize the error function (24) to within some small fluctuation range. The latter expectation is based on the RFALCON's potential as a universal approximator.

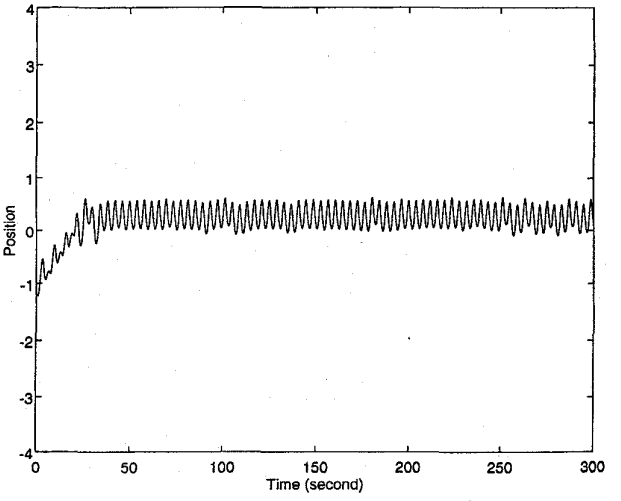


Fig. 28. Position deviation of the ball allowed by a trained RFALCON when ball mass is doubled to 0.1 kg from 0.05 kg.

Reinforcement learning has been widely used to solve various practical problems. The peg-in-hole insertion problem is difficult to model analytically and exemplifies the difficulties raised by uncertainty in real-world control problems. This task is also highly relevant to industrial robotics because about 33% of all automated assembly operations are peg-in-hole insertions, which makes them the most common assembly operation. The abstract peg-in-hole task can be solved quite easily if the exact location of the hole is known and if the manipulator can precisely control the position and orientation of the peg. Real-world conditions, however, present uncertainties due to errors and noise in sensory feedback, errors in execution of motion command and movement of the part grasped by the robot, and substantially degrade the performance of conventional position control methods. The success of the direct reinforcement learning approach [34] to training the controller indicates that this approach can be useful for automatically synthesizing robot control strategies that satisfy constraints encoded in the performance evaluations. Another important problem in robotics is that of generating paths between initial and goal positions that

avoids collisions with obstacles. In practice, the robots have incomplete information about environments, knowing only their own sensory data. This kind of problem cannot be modeled exactly and explicitly because many possible paths between initial and goal positions exist. A reinforcement-based connectionist system that enabled a mobile robot to find and learn obstacle-avoiding paths in a nonmaze-like two-dimensional environment was presented in [39]. We are also using the proposed RFALCON model to solve many practical problems including crane position control and adaptive control of electro-discharge machining (EDM).

VI. Conclusion

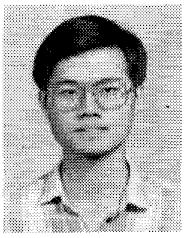
This paper describes an RFALCON for solving various reinforcement learning problems. By combining temporal difference techniques, stochastic exploration, and a proposed on-line supervised structure/parameter-learning algorithm, a reinforcement structure/parameter-learning algorithm was derived for the RFALCON. Using the proposed connectionist structure and learning algorithm, a fuzzy logic controller that controls a plant and a fuzzy predictor that models the plant can be set up dynamically through simultaneous structure/parameter-learning for reinforcement learning problems. The proposed RFALCON makes the design of fuzzy logic controllers more practical for real-world applications, since it greatly lessens the quality and quantity requirements of the teaching signals. More importantly, it reduces the combinatorial demands placed by the standard methods for adaptive linearization of input–output spaces in existing fuzzy control systems. This makes possible the use of fuzzy controllers in real-world complex systems that involve many state/control variables. Computer simulations of the cart-pole balancing and the ball and beam problems satisfactorily verified the validity and performance of the proposed RFALCON. Future work will focus on applying the RFALCON to practical problems in the real world.

Acknowledgment

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

References

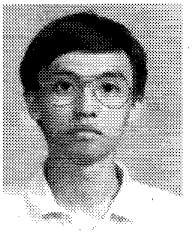
[1] G. E. Hinton, "Connectionist learning procedures," *Artificial Intell.*, vol. 40, no. 1, pp. 143–150, 1989.
[2] C. J. Lin and C. T. Lin, "An ART-based fuzzy adaptive learning control network," in *Proc. IEEE Int. Conf. on Fuzzy Systems*, 1994, pp. 1–6.
[3] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Development*, vol. 3, pp. 221–229, 1959.
[4] A. Dickinson, *Contemporary Animal Learning Theory.* Cambridge, U.K.: Cambridge Univ. Press, 1980.
[5] E. R. Hilgard and G. H. Bower, *Theories of Learning*, 4th ed. Englewood Cliffs, NJ: Prentice-Hall, 1975.
[6] W. K. Estes, "Toward a statistical theory of learning," *Psych. Rev.*, vol. 57, pp. 94–107, 1950.
[7] M. L. Tsetlin, *Automation Theory and Modeling of Biological Systems.* New York: Academic, 1973.
[8] T. M. Cover and M. E. Hellman, "The two-armed bandit problem with time-invariant binite memory," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 185–195, 1970.
[9] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction.* Englewood Cliffs, NJ: Prentice-Hall, 1989.
[10] A. H. Klopf, *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence.* Washington, DC: Hemisphere, 1982.
[11] A. G. Barto and R. S. Sutton, "Landmark learning: An illustration of associative search," *Biol. Cybern.* vol. 42, pp. 1–8, 1981.
[12] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problem," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 834–847, 1983.
[13] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, Univ. Massachusetts, Amherst, 1984.
[14] 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.
[15] A. G. Barto and M. I. Jordan, "Gradient following without backpropagation in layered network," in *Proc. Int. Joint Conf. Neural Networks*, vol. II, San Diego, CA, pp. 629–636.
[16] R. S. Sutton, "Learning to predict by the methods of temporal difference," *Machine Learning*, vol. 3, pp. 9–44, 1988.
[17] R. J. Williams, "A class of gradient-estimating algorithms for reinforcement learning in neural networks," in *Proc. Int. Joint Conf. Neural Networks*, vol. II, San Diego, CA, 1987, pp. 601–608.
[18] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, pp. 724–740, 1992.
[19] C. W. Anderson, "Strategy learning with multilayer connectionist representations," in *Proc. 4th Int. Wkshp. Mach. Learning*, Irvine, CA, June 1987, pp. 103–114.
[20] C. T. Lin and C. S. G. Lee, "Reinforcement structure/parameter learning for neural network-based fuzzy logic control systems," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 1993, pp. 88–93.
[21] ——, "Neural network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol.C-40, pp. 1320–1336, 1991.
[22] ——, "Real-time supervised structure/parameter learning for fuzzy neural network," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 1992, pp. 1283–1290.
[23] C. T. Lin, C. J. Lin, and C. S. G. Lee, "Fuzzy adaptive learning control network with on-line neural learning," to appear in *Fuzzy Sets Syst.*
[24] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.
[25] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. Neural Networks*, vol. 3, pp. 698–712, 1992.
[26] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Parts I and II," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-20, pp. 404–435, 1990.
[27] B. Kosko, *Neural Networks And Fuzzy Systems.* Englewood Cliffs, NJ: Prentice-Hall, 1992.
[28] C. C. Lee and H. R. Berenji, "An Intelligent controller based on approximate reasoning and reinforcement learning," in *Proc. IEEE Int. Symp. Intell. Contr.*, 1989, pp. 200–205.
[29] 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., ch.3. Cambridge, MA: MIT Press, 1990.
[30] D. Michie and R. A. Chambers, "BOXES: An experiment in adaptive control," in *Machine Intelligence*, E. Dale and D. Michie, Eds., vol. 2. Edinburgh, Scotland: Oliver and Boyd, 1968, pp. 137–152.
[31] P. K. Simpson, "Fuzzy min-max neural networks—Part 2: Clustering," *IEEE Trans. Fuzzy Systems*, vol. 1, 1993, pp. 32–45.
[32] J. A. Franklin, "Input space representation for reinforcement learning control," in *Proc. IEEE Int. Symp. Intell. Contr.*, 1989, pp. 115–122.
[33] H. Benbrahim, J. S. Doleac, J. A. Franklin, and O. G. Selfridge, "Real-time learning: A ball on a beam," in *Proc. Int. Joint Conf. Neural Networks*, vol. I, 1992, pp. 98–103.
[34] V. Gullapalli, J. A. Franklin, and H. Benbrahim, "Acquiring robot skills via reinforcement learning," *IEEE Contr. Syst. Mag.*, pp. 13–24, 1994.
[35] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions," *Neural Networks*, vol. 3, pp. 671–692, 1990.
[36] J. Hauser, S. Sastry, and P. Kokotovic, "Nonlinear control via approximate input–output linearization: The ball and beam example," *IEEE Trans. Automat. Contr.*, vol. 37, pp. 392–398, 1992.
[37] L. X. Wang and J. M. Mendel, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks*, vol. 3, 1992, pp. 807–814.
[38] W. Rudin, *Principles of Mathematical Analysis.* New York: McGraw-Hill, 1976.
[39] J. del R. Millán and C. Torras, "Connectionist approaches to robot path finding," in *Progress in Neural Networks Series*, O. M. Omidvar, Ed., vol. 3. Norwood, NJ: Ablex, 1991.



Cheng-Jian Lin received the B.S. degree in electrical engineering from Tatung Institute of Technology, Taiwan, R.O.C., in 1986 and the M.S. degree in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1991.

Currently, he is a candidate for the Ph.D. degree in the Department of Control Engineering, the National Chiao-Tung University, Taiwan, R.O.C. His current research interests are neural networks, learning systems, fuzzy control, approximate reasoning, and signal processing.

Mr. Lin is a student member of the IEEE Control Systems Society.



Chin-Teng Lin (S'88–M'91) received the B.S. degree in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1986, and the M.S.E.E. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN, in 1989 and 1992, respectively.

Since August 1992, he has been with the College of Electrical Engineering and Computer Science, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., where he is currently an Associate Professor of Control Engineering. His current research interests are fuzzy systems, neural networks, intelligent control, human-machine interface, and video and audio processing. He is the author of *Neural Fuzzy Control Systems with Structure and Parameter Learning* (World Scientific).

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.