1

Self-Learning Fuzzy Controllers Based on Temporal Back Propagation

Jyh-Shing R. Jang
Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720
jang@diva.berkeley.edu

Abstract

This paper presents a generalized control strategy that enhances fuzzy controllers with self-learning capability for achieving prescribed control objectives in a near-optimal manner. This methodology, termed temporal back propagation, is model-insensitive in the sense that it can deal with plants that can be represented in a piecewise differentiable format, such as difference equations, neural networks, GMDH, fuzzy models, etc. Regardless of the numbers of inputs and outputs of the plants under consideration, the proposed approach can either refine the fuzzy if-then rules obtained from human experts, or automatically derive the fuzzy if-then rules if human experts are not available. The inverted pendulum system is employed as a testbed to demonstrate the effectiveness of the proposed control scheme and the robustness of the acquired fuzzy controller.

1 Introduction

Fuzzy controllers (FC's) have recently found various applications in industry as well as in household appliances. For complex and/or ill-defined systems that are not easily controlled by conventional control schemes, FC's provides a feasible alternative since they can easily capture the approximate, qualitative aspects of human knowledge and reasoning. However, the performance of FC's relies

*Research supported in part by NASA Grant NCC-2-275; LLNL Grant No. ISCR 89-12; MICRO State Program Award No. 90-191; MICRO Industry: Rockwell Grant No. B02302532.

on two important factors: the soundness of knowledge acquisition techniques and the availability of domain (human) experts. These two factors substantially restrict the application domains of FC's.

We have proposed the ANFIS (Adaptive-Network-based Fuzzy Inference System) architecture [7, 6, 8] to solve the first problem concerning the automatic elicitation of knowledge in the forms of fuzzy if-then rules. The proposed architecture can identify the near-optimal membership functions and other parameters of a rule base for achieving a desired input-output mapping. The basics of the ANFIS architecture are introduced in the next section.

This paper addresses the second problem: how to control a system through a self-learning FC? In other words, without resorting to human experts, we want to construct a FC that can perform a prescribed control task. The learning aspects of FC's have always been a interesting topic, and recent developments are mostly based on reinforcement learning [9, 10, 2]. Our learning method is based on a special form of gradient descent (called back propagation), which is used for training artificial neural networks [13, 15]. To control the plant's trajectory, we apply the back-propagation-type gradient descent method to propagate the error signals through different time stages. This is called TBP (*temporal back propagation*) and it is explained in Section 3.

The proposed control strategy is quite general and can be used to control plants with diverse characteristics. Moreover, the *a prori* knowledge that we have about the plant can be applied in an auxiliary manner to speed up the learning process. In our simulation described in Section 4, we successfully employ the TBP to construct a fuzzy controller with only 4 fuzzy if-then rules for balancing an inverted pendulum system. The discussion and conclusions are given in Section 5.

2 Basics of ANFIS

When employed as a controller, a *fuzzy inference system* is often called a fuzzy controller; therefore we will use the terms fuzzy inference system and fuzzy controller interchangeably throughout this paper. In this section, we describe the basics of *adaptive networks* which include artificial neural networks as a special case. A special configuration of adaptive networks, called *ANFIS* (*Adaptive-Network-based Fuzzy Inference System*), is introduced in detail since it is functionally equivalent to a fuzzy controller.

A. Adaptive Networks

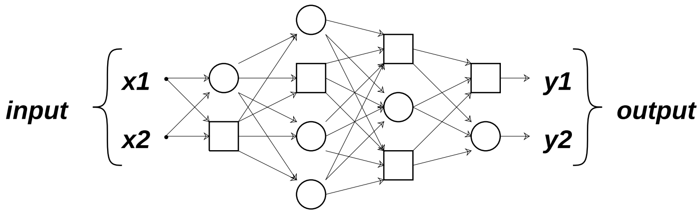

Figure 1: *An adaptive network.*

An adaptive network (Figure 1) is a multi-layer feedforward network in which each node performs a particular function (*node function*) on incoming signals using a set of parameters specific to this node. The form of node functions may vary from node to node, and the choice of each node function depends on the overall function which the adaptive network is designed to implement.

To reflect different adaptive capabilities, we use both circle and square nodes in an adaptive network. A square node (adaptive node) has modifiable parameters while a circle node (fixed node) has none. The parameter set of an adaptive network is the union of the parameter sets of each adaptive node. In order to achieve a desired input-output mapping, these parameters are updated according to given training data and a gradient-based update procedure described below.

Suppose that a given adaptive network has L layers and the k-th layer has $\#(k)$ nodes. We can denote the node in the i-th position of the k-th layer by (k, i), and its node function (or node output) by O_i^k. Since a node output depends on its incoming signals and its parameter set, we have

$$O_i^k = O_i^k(O_1^{k-1}, \ldots O_{\#(k-1)}^{k-1}, a, b, c, \ldots), \tag{1}$$

where a, b, c, etc. are the parameters pertaining to this node. (Note that we use O_i^k as both the node output and node function.)

Assuming the given training data set has P entries, we can define the *error measure* for the p-th ($1 \leq p \leq P$) entry of training data entry as the sum of squared errors:

$$E_p = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2, \tag{2}$$

where $T_{m,p}$ is the m-th component of p-th target output vector, and $O_{m,p}^L$ is the m-th component of an actual output vector produced by the presentation of the p-th input vector. Hence the overall error measure is $E = \sum_{p=1}^{P} E_p$.

In order to develop a update procedure that implements gradient descent in E over the parameter space, first we have to calculate the *error signal* $\frac{\partial E_p}{\partial O}$ for p-th training data and for each node output O. The error signal for the output node at (L, i) can be calculated readily from equation (2):

$$\frac{\partial E_p}{\partial O_{i,p}^L} = -2(T_{i,p} - O_{i,p}^L). \tag{3}$$

For the internal node at (k, i), the error signal can be derived by the chain rule:

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k}, \tag{4}$$

where $1 \leq k \leq L-1$. That is, the error signal of an internal node can be expressed as the linear combination of the error signals of the nodes in the next layer. Therefore for all $1 \leq k \leq L$ and $1 \leq i \leq \#(k)$, we can find $\frac{\partial E_p}{\partial O_{i,p}^k}$ by equation (3) and (4).

Now if α is a parameter of the given adaptive network, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha}, \tag{5}$$

where S is the set of nodes whose outputs depend on α. Then the derivative of the overall error measure E with respect to α is

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha}. \tag{6}$$

Accordingly, the update formula for the generic parameter α is

$$\triangle \alpha = -\eta \frac{\partial E}{\partial \alpha}, \tag{7}$$

in which η is a update rate (or learning rate) of parameter α. Usually η is further expressed as

$$\eta = \frac{S}{\sqrt{\sum_{\alpha} (\frac{\partial E}{\partial \alpha})^2}}, \tag{8}$$

where S is the *step size*, the length of each gradient transition in parameter space. By a proper selection of S, we can vary the speed of convergence.

B. ANFIS: Adaptive-Network-Based Fuzzy Inference System

For simplicity, we assume the fuzzy inference system under consideration has two inputs x and y, one output z, and the rule base contains two fuzzy if-then rules of Takagi and Sugeno's type [14]. The corresponding ANFIS architecture is shown in Figure 2 where node functions in the same layer are of the same type, as described below:

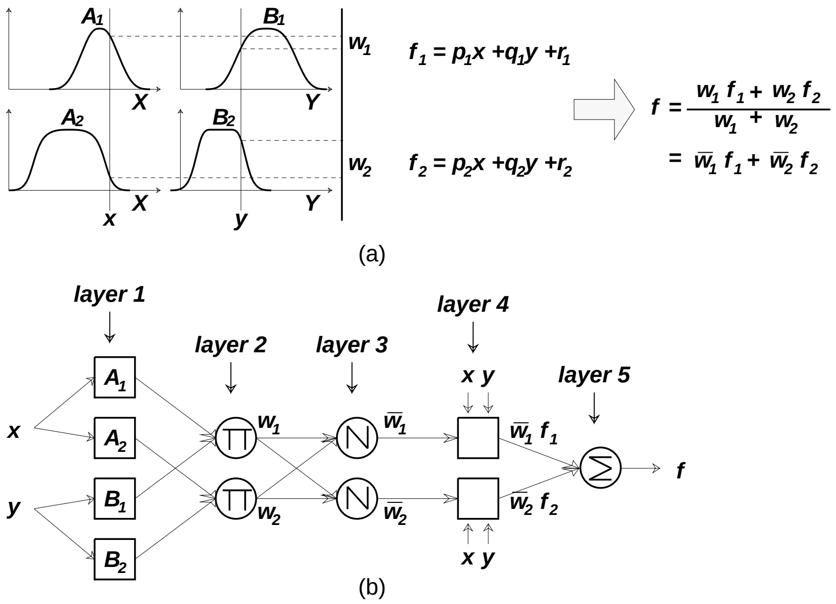

Figure 2: *(a) Fuzzy inference system; (b) equivalent ANFIS.*

Layer 1 Every node i in this layer is a square node with a node function

$$O_i^1 = \mu_{A_i}(x), \tag{9}$$

where x is the input to node i, A_i is the linguistic label (*small* , *large*, etc.) associated with this node function. In other words, O_i^1 is the membership function of A_i and it specifies the degree to which the given x satisfies the quantifier A_i. Usually we choose $\mu_{A_i}(x)$ to be bell-shaped with maximum equal to 1 and minimum equal to 0, such as

$$\mu_{A_i}(x) = \frac{1}{1 + [(\frac{x - c_i}{a_i})^2]^{b_i}}, \tag{10}$$

or

$$\mu_{A_i}(x) = exp\{-[(\frac{x - c_i}{a_i})^2]^{b_i}\}, \tag{11}$$

where $\{a_i,\, b_i,\, c_i\}$ is the parameter set. As the values of these parameters change, the bell-shaped functions vary accordingly, thus exhibiting various forms of membership functions on linguistic label A. In fact, any continuous and piecewise differentiable functions, such as commonly used trapezoidal or trangular-shaped membership functions, are also qualified candidates for node functions in this layer. Parameters in this layer are referred to as *premise parameters*.

Layer 2 Every node in this layer is a circle node labeled Π which multiplies the incoming signals and sends the product out. Each node output represents the *firing strength* (or *weight*) of a rule. (In fact, other *T-norm* operators that perform generalized AND can be used as the node function for this layer.)

Layer 3 Every node in this layer is a circle node labeled N. The i-th node calculates the ratio of the i-th rule's firing strength to the sum of all rules' firing strengths. In other words, nodes in this layer compute the *normalized firing strength* of each rule.

Layer 4 Every node i in this layer is a square node with a node function

$$O_i^4 = \bar{w}_i(p_i x + q_i y + r_i), \tag{12}$$

where $\bar{w}_i$ is the output of layer 3, and $\{p_i,\, q_i,\, r_i\}$ is the parameter set. Parameters in this layer are referred to as *consequent parameters*.

Layer 5 It's a circle node labeled Σ that sums all incoming signals.

Thus we have constructed an adaptive network which is functionally equivalent to a fuzzy inference system. This ANFIS architecture then can update its parameters according to the gradient descent update procedure mentioned before. Other ANFIS's corresponding to different type of fuzzy if-then rules and defuzzification mechanisms, plus a fast update procedure combining gradient method and least square estimate, can be found in [6, 8].

3 Constructing Self-Learning Fuzzy Controllers

In this section, we propose a generalized control scheme which can construct a fuzzy controller through *temporal back propagation*, such that the state variables can follow a given desired trajectory as close as possible. The basic idea is to implement both the controller and the plant at each time stage as a *stage adaptive network*, and cascade these stage adaptive networks into a *trajectory adaptive network* to facilitate the temporal back propagation learning process.

A. Stage Adaptive Network

Figure 3 illustrates the block diagram of a feedback control system consisting of a fuzzy controller and a plant. We assume the delay through the controller is small and the state variables are accessible with accuracy. Moreover, the plant block is viewed as a static system since the

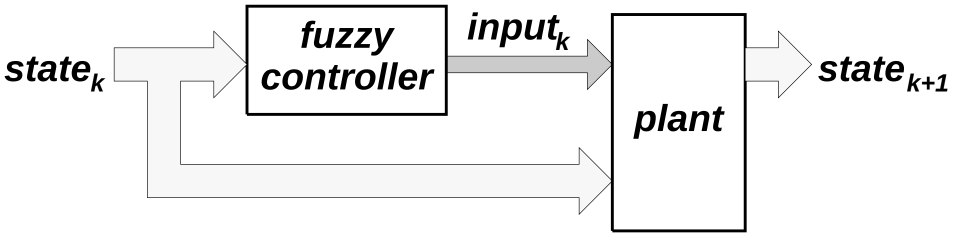

Figure 3: *Block diagram of a fuzzy controller and a plant. Also a stage adaptive network at time stage k.*

dependency of the next state on the present state is shown explicitly. Before finding a controller to control the plant state, we have to find mathematical expressions for both the controller block and the plant block. This step is referred to as the *implementation*. In our case, we are going to implement both blocks as adaptive networks.

An obvious candidate for implementing the FC block in Figure 3 is the ANFIS architecture since it has exactly the same function as a fuzzy controller, as shown in Figure 2. If we have p inputs to the plant, then the FC block can be implemented either as p ANFIS's, or as an ANFIS that has rules with multiple consequents.

Suppose that we have a human expert who knows how to control the plant. Then the domain knowledge can be transformed into fuzzy if-then rules and the corresponding parameters (which characterize membership functions) can be used as the initial parameters of the FC block in the learning process. As a result, the domain knowledge can guide the TBP learning process to get started from a point in the parameter space that is not far from the optimal one, and the TBP can fine-tune the domain knowledge for achieving a better performance. This cooperative relation between the domain knowledge and the TBP learning process is not always present in other types of controllers.

On the other hand, if we don't have *a prori* knowledge about controlling the plant, then the number of fuzzy if-then rules has to be decided more or less by trial and error. Fortunately, due to ANFIS's remarkable representational power [6, 8], usually we don't need many rules to construct the desired mapping from state variables to control action.

As for the implementation of the plant block, we can choose whatever function approximators that can best represent the input-output behavior of the plant. This model-insensitive attribute is mostly due to the flexibility of adaptive networks, which allows us to choose either conventional models (difference or differential equations, transfer functions, etc) or unconventional ones (ANFIS, neural networks [13], radial basis function networks [11], GMDH structure [5], etc.) to implement the plant block.

In case the plant can be modeled as a set of n (= number of state variables) first-order difference equations, then the plant block can be replaced with n nodes, each of which uses one difference equation to get the state variable at the next time step. Furthermore, if the state equations of the plant are a set of first-order differential equations:

$$\dot{\vec{x}}(t) = \vec{f}(\vec{x}(t),\ \vec{in}(t),\ t), \tag{13}$$

where $\vec{x}(t)$ is a vector consisting of state variables at time t and $\vec{in}(t)$ is the input vector to the plant, then we can just employ a linear approximation to get the difference equations as below

$$\vec{x}(h*k+h) = h*\vec{f}(\vec{x}(h*k),\ \vec{in}(h*k),\ h*k) + \vec{x}(h*k), \tag{14}$$

where k is an integer and h is the sampling time. Therefore the plant block still has n node, each of which performs a component function of equation (14).

When the sampling time h is too big or the plant has fast dynamics, the linear approximation may not be a reasonable estimate of the next state. In this case, we can utilize a large body of numerical analysis techniques to obtain a more precise estimate, for instance, the second-order Runge-Kutta method:

$$\begin{cases} \vec{a} = h*\vec{f}(\vec{x}(h*k),\ \vec{in}(h*k),\ h*k), \\ \vec{b} = h*\vec{f}(\vec{x}(h*k)+\vec{a},\ \vec{in}(h*k),\ h*k+h), \\ \vec{x}(h*k+h) = \vec{x}(h*k) + 0.5*(\vec{a}+\vec{b}). \end{cases} \tag{15}$$

However, to implement the above equations as an adaptive network (without modifiable parameters) is more complex and some intermediate nodes would present in the resulting network for the intermediate variable vector $\vec{a}$ and $\vec{b}$. Higher order Runge-Kutta formulas may be used to implement the plant block as well, but the increased complexity of the adaptive network could slow down the learning process even more.

Consequently, the block diagram of Figure 3 can also be viewed as an adaptive network containing two sub-networks, the FC block (ANFIS) and the plant block. Subsequently, we refer to the adaptive network of Figure 3 as SAN_k, representing the *stage adaptive network* at time stage k.

B. Trajectory Adaptive Network

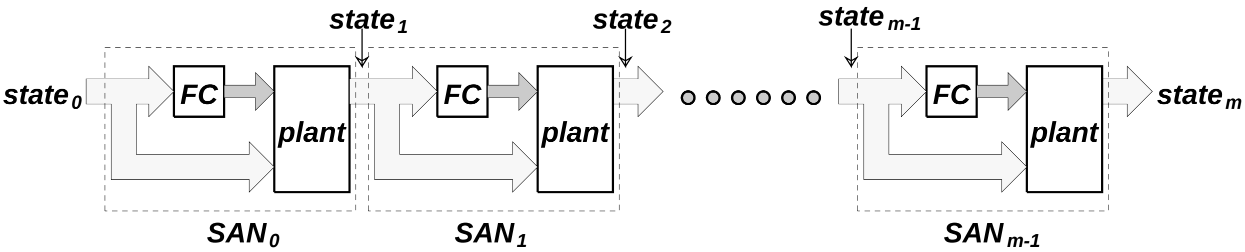

Figure 4: *State transition diagram.*

Given the state of the plant at time $t = k * h$, the FC will generate an input to the plant and the plant will evolve to the next state at time $(k+1) * h$. By repeating this process starting from $t = 0$, we obtain a plant state trajectory determined by the initial state and the parameters of the FC. The state transition from $t = 0$ to $t = m * h$ is shown conceptually in Figure 4 which is, again, an adaptive network consisting of m SAN_k, $k = 0$ to $m-1$. Accordingly we can still apply the back-propagation gradient descent to minimize the differences between adaptive network outputs and desired outputs. In order to make the inputs and outputs more explicit, we redraw Figure 4 to get the *trajectory adaptive network* shown in Figure 5, where the inputs to the network are the initial state of the plant at time = 0; the outputs of the network are the state trajectory from $t = h$ to $m * h$; and the adjustable parameters are all pertaining to the FC block implemented as an ANFIS. Hence each entry of the training data is of the form

$$(initial\ state;\ desired\, trajectory), \tag{16}$$

and the corresponding error measure to be minimize is

$$E = \sum_{k=1}^{m} ||\vec{x}(h*k) - \vec{x}_d(h*k)||^2 \tag{17}$$

where $\vec{x}_d(h*k)$ is the desired trajectory at $t = h*k$. With some minor modifications of Figure 5, the above error measure can be revised as

$$E = \sum_{k=1}^{m} ||\vec{x}(h*k) - \vec{x}_d(h*k)||^2 + \lambda * \sum_{k=0}^{m-1} ||\vec{in}(h*k)||^2 \tag{18}$$

where $\vec{in}(h*k)$ is the controller's output at time $h*k$. By a proper selection of λ, a compromise between trajectory error and control effort can be obtained.

Since the error signals of back-propagation can propagate through different time stages, this control methodology is called *temporal back-propagation*, or simply *TBP*. As a matter of fact, the basic idea of TBP is similar to Nguyen and Widrow's approach [12] to construct a self-learning

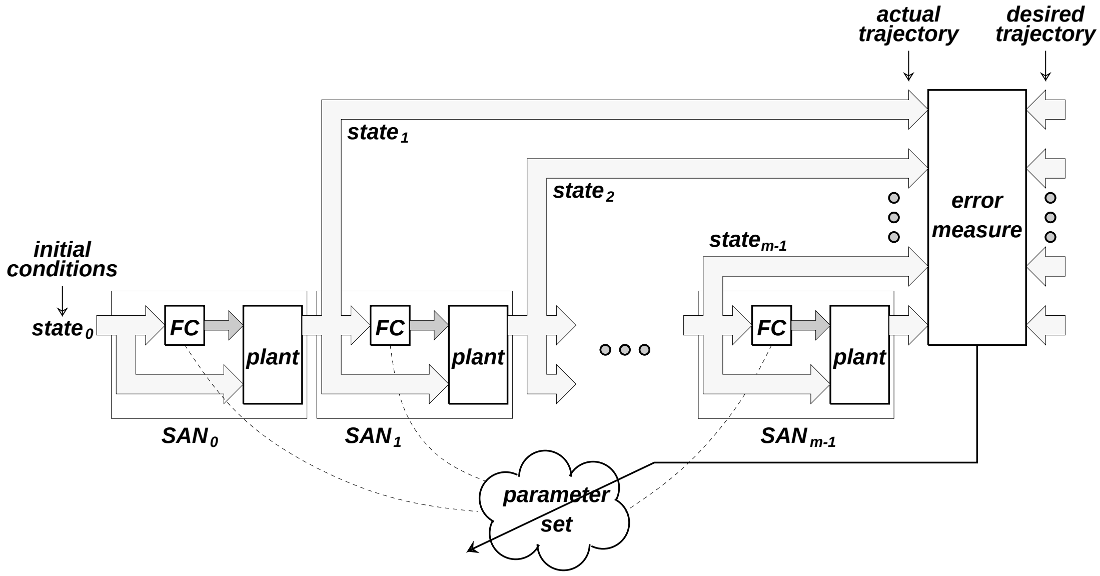

Figure 5: *A trajectory adaptive network for control application.*

neural controller, which is called back-propagation through time by Werbos [4]. We generalize the idea to a much more flexible building block, the adaptive network, which has two advantages over neural networks:

1. Accomodation of *a prori* knowledge from human operator, in the form of fuzzy if-then rules.

2. No need to remodel the plant in neural networks if we already have other existing model for it, such as difference equations.

In the trajectory adaptive network shown in Figure 5, though there are m FC blocks, all of them refer to the same fuzzy controller at different time stages. Namely, there is only one parameter set which belongs to all m FC blocks at different time stages. For clarity, this parameter set is shown explicitly in Figure 5 and it is updated according to the output of the error measure block.

4 Application to the Inverted Pendulum System

The proposed control scheme is quite general and it can be applied to a variety of control problems. In this section, we demonstrate the effectiveness of the TBP by applying it to a benchmark problem in intelligent control — the inverted pendulum system.

A. The Inverted Pendulum System

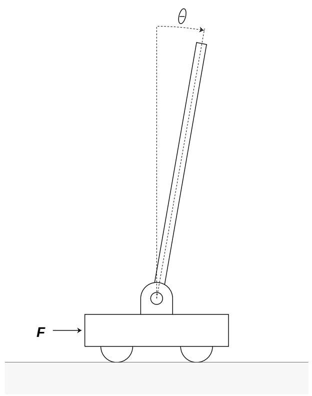

Figure 6: *The inverted pendulum system.*

The inverted pendulum system (Figure 6) is composed of a rigid pole and a cart on which the pole is hinged. The cart moves on the rail tracks to its right or left, depending on the force exerted on the cart. The pole is hinged to the cart through a frictionless free joint such that it has only one degree of freedom. The control goal is to balance the pole starting from nonzero conditions by supplying appropriate force to the cart.

The dynamics of the inverted pendulum system are characterized by four state variables: θ (angle of the pole with respect to the vertical axis), $\dot{\theta}$ (angular velocity of the pole), z (position of the cart on the track) and $\dot{z}$ (velocity of the cart). The behavior of these four state variables is governed by the following two second-order differential equations [3, 1]:

$$\ddot{\theta} = \frac{g * sin\theta + cos\theta * \left(\frac{-F - m * l * \dot{\theta}^2 * sin\theta}{m_c + m}\right)}{l * \left(\frac{4}{3} - \frac{m * cos^2\theta}{m_c + m}\right)}, \tag{19}$$

$$\ddot{z} = \frac{F + m * l * (\dot{\theta}^2 * sin\theta - \ddot{\theta} * cos\theta)}{m_c + m}, \tag{20}$$

where g (acceleration due to gravity) is 9.8 $meter/sec^2$, m_c (mass of cart) is 1.0 kg, m (mass of pole) is 0.1 kg, l (half length of pole) is 0.5 m, and F is the applied force in newtons. Our control goal here is to balance the pole without regard to the cart's position and velocity, hence only equation (19) is relevant in our simulation.

B. Simulation Settings

Figure 7 shows the stage adaptive network used in our simulation. Both the controller and the plant block, together with the learning rule, are explained below.

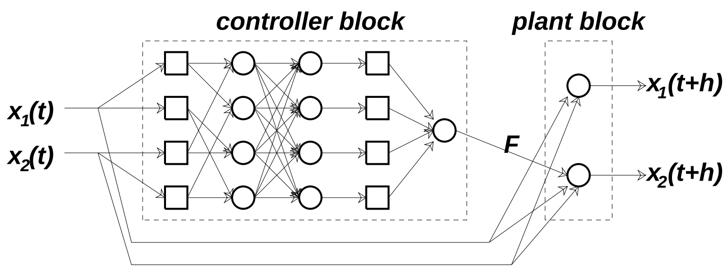

Figure 7: *The implementation of a stage adaptive network.*

Plant Block

As mentioned earlier, there are several ways to implement the plant block depending on how well we know the plant. In this case, the plant is a deterministic nonlinear dynamical system with precisely defined differential equations, so we can just use 2 nodes to calculate the state variables at the next time step by linear approximation

$$\begin{cases} x_1(t+h) &= h\dot{x}_1(t) + x_1(t), \\ x_2(t+h) &= h\dot{x}_2(t) + x_2(t), \end{cases} \tag{21}$$

where $x_1(\cdot) = \theta(\cdot)$, $x_2(\cdot) = \dot{\theta}(\cdot)$. These two equations are the node functions of the plant block in Figure 7.

Controller Block

We assume no domain knowledge (from a human operator's point of view) about the inverted pendulum system is available. The controller block in Figure 3 is implemented as an ANFIS with two inputs, each of which is assigned two membership functions, so it is a fuzzy controller with 4 fuzzy if-then rules of Takagi and Sugeno's type [14]. See the controller block in Figure 7. (Though the number of fuzzy rules can be more than four, the simulation indicates 4 rules are enough for balancing the pole.)

Without any domain knowledge, we have to set the initial parameters subjectively. The consequent parameters of the FC are all set at zeros, which means the control action is zero initially as shown in Figure 9. As a conventional way of setting membership functions in a fuzzy controller, the premise parameters are set in such a way that the membership functions can cover the domain interval (or universe of discourse) completely with sufficient overlapping of each other. Figure 8(a) and (b) illustrates the initial membership functions in the form of equation (10); the domain interval for θ (degrees) and $\dot{\theta}$ (degrees/sec) are assumed to be [-20, 20] and [-50, 50], respectively.

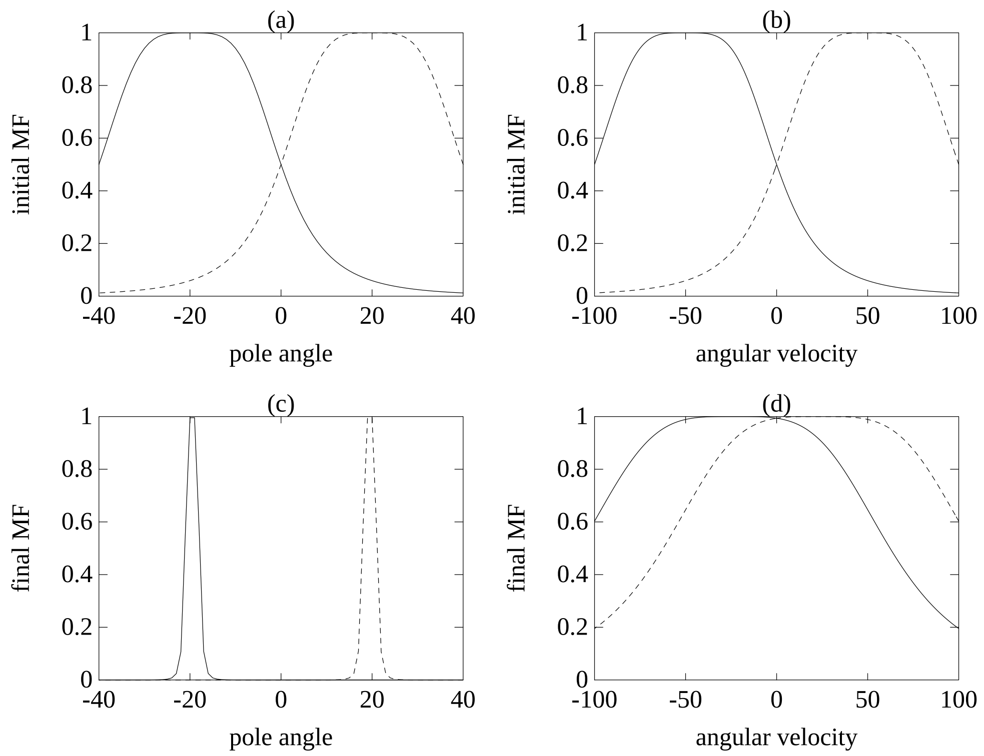

Figure 8: *(a)(b) Initial membership functions; (c)(d) final membership functions..*

Temporal Back Propagation

We employ 100 stage adaptive networks to construct the trajectory adaptive network, and each stage adaptive network corresponds to the time transition of 10 ms. That is, the time step (h) used is 10 ms, and the trajectory adaptive network corresponds to a time interval from $t = 0$ to $t = 1$ sec. If h is too small, a large network has to be built to cover the same time span, which increases the signal propagation time and thus delays the whole learning process. On the other hand, if h is too big, then the the linear approximation of the plant behavior may not be precise enough and a higher order approximation has to be used instead.

The training data set contains desired input-output pairs of the format

$$(initial\ condition;\ desired\ trajectory), \tag{22}$$

where the initial condition is a two-element vector which specifies the initial condition of the pole; the desired trajectory is a 100-element vector which contains the desired pole angle at each time step. In our simulation, only 2 entries of training data are used: the initial conditions are $(10, 0)$ and $(-10, 0)$, respectively, and the desired trajectory is always a zero vector. In short, we expect that the trajectory adaptive network can not only learn to balance the pole from an initial pole angle of $+10$ or -10 degrees, but also achieve the control goal in an near-optimal manner which minimizes the error measure

$$E = \sum_{k=1}^{100} \theta^2(0.01 * k) + \lambda * \sum_{k=0}^{99} f^2(0.01 * k), \tag{23}$$

where $f(0.01 * k)$ is the controller's output force and λ $(= 10)$ accounts for the relative unit cost of control effort.

To speed up the convergence, we follow a strict gradient descent in the sense that each transition of the parameters will lead to a smaller error measure. If the error measure increases after parameter update, we back up to the original point in the parameter space and decrease the current step size by half. This process is repeated until the weight update leads to a smaller error measure. However, this step size update rule tends to use a small step size if the error measure surface encountered in the first few updates is not smooth. Therefore we multiply the step size by 4 after observing 3 consecutive transitions without any back-up actions. The initial step size in the simulation is 20 and the learning process stops whenever the number of transitions in parameter space (which is equal to the number of reductions in error measure) reaches 10.

C. Simulation Results

All the simulation settings mentioned above are referred to as the *reference setting*; other simulations are based on this setting with minor changes. In the learning task with the reference setting, it is amazing to observe that the FC is able to balance the pole right after the first parameter transition, and it keeps on refining the controller (minimizing the error measure) till the 10-th parameter transition is done. Figure 8 (a) and (b) shows the initial membership functions on pole angle and angular velocity, (c) and (d) shows the final membership functions. Figure 9 is the initial control action surface, Figure 10 is the final control action surface after 10-th parameter transition. The listing of the fuzzy rules with numerical parameters can be found in the Appendix.

Figure 8 indicates that the final membership functions for θ are quite different from the initial membership functions. Visually there are no membership functions covering the interval [-10, 10] of θ, making the linguistic interpretation of the fuzzy rules difficult. However, since we are utilizing the FC as a functional approximator that can generate the required nonlinear mapping,

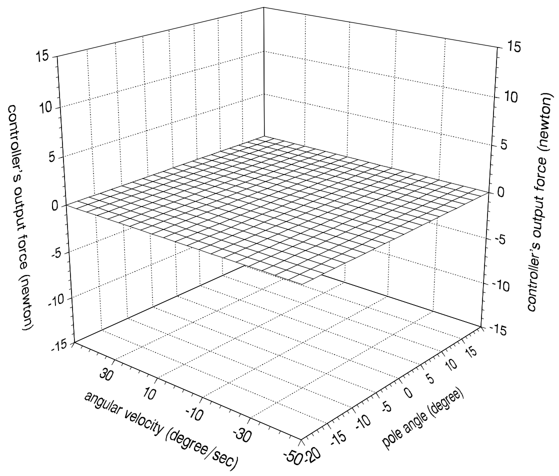

Figure 9: *Initial control action surface.*

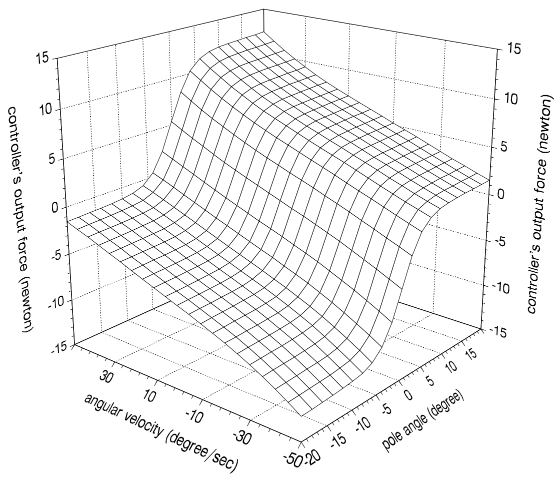

Figure 10: *Final control action surface.*

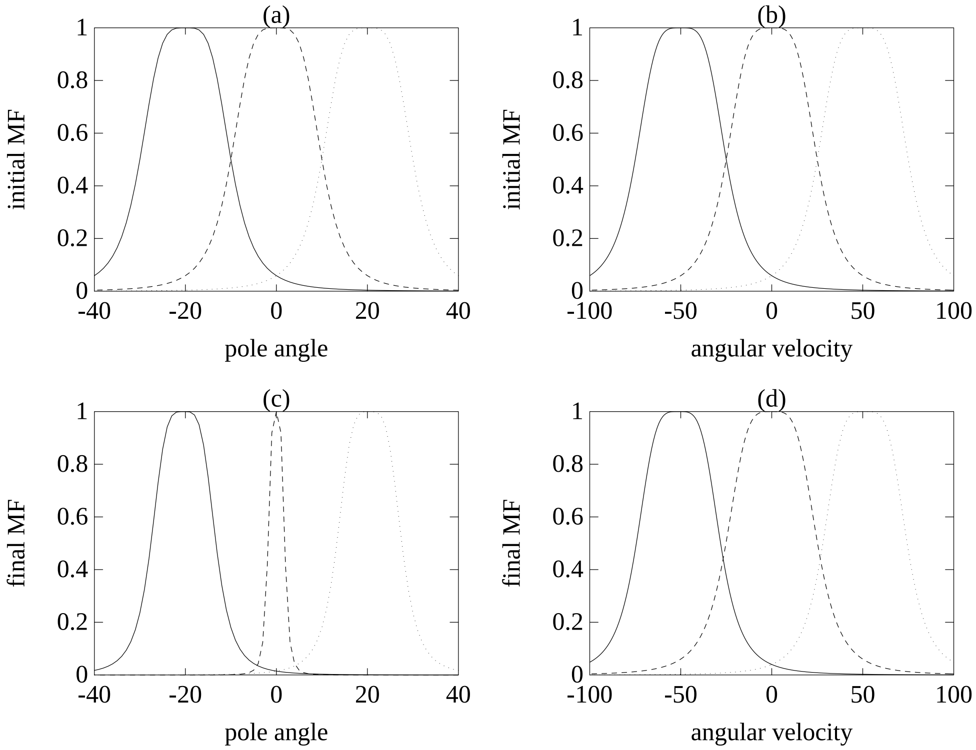

Figure 11: *(a)(b) Initial membership functions and (c)(d) final membership functions of a 9-rule fuzzy controller.*

linguistically desirable features (such as enough overlapping between membership functions and total coverage of the domain interval) don't have to be one of the FC's attributes in this case. If we want to keep those desirable feature, we can either impose some constraints on the premise parameters, or simply increase the number of parameters of the fuzzy controller to endow it with more degrees of freedom. Figure 11 shows the membership functions of a 9-rule fuzzy controller which has about the same performance as the 4-rule fuzzy controller. Due to its higher degrees of freedom, the premise parameters of the 9-rule fuzzy controller don't have to change a lot to minimize the error measure, therefore the the final membership functions can cover all the domain intervals with desirable overlapping.

Solid curves in Figure 12 demonstrates the state variable trajectories under the reference setting: (a), (b) and (d) show the pole angle (degree), angular velocity (degree/sec) and control actions (Newtons) from $t = 0$ to $t = 2$ *sec*; (c) is the state space plot which reveals how the trajectory approaches the origin from the initial point (10, 0). Dashed and dotted curves in

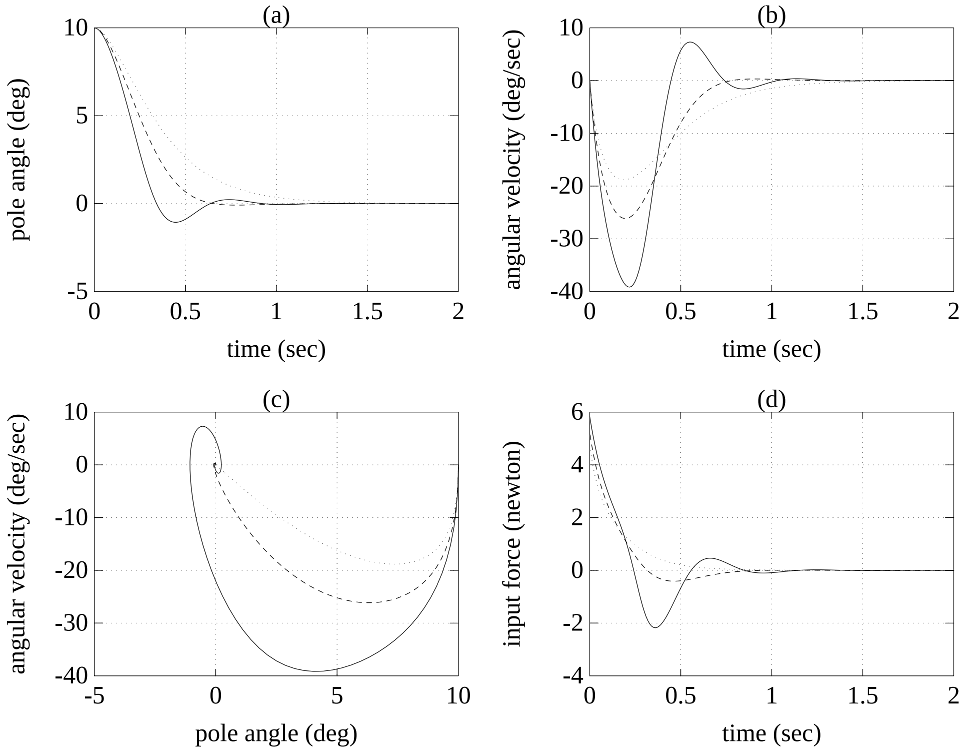

Figure 12: *(a) Pole angle; (b) pole angular velocity; (c) state space; (d) input force. (Solid, dashed and dotted curves correspond to λ = 10, 40 and 100, respectively.)*

Figure 12 correspond to λ equal to 40 and 100, respectively. From (a), it is observed that a smaller λ (solid curve) achieves the control goal faster since the controller can apply a larger force to balance the pole. For a large λ (dotted curve), the controller's output has to be kept small, thus slowing down the approach to the goal.

To demonstrate how the fuzzy controller can survive substantial changes of plant parameters, we use poles of different lengths to test the controller obtained from the reference setting. The results are shown in Figure 13, where solid, dashed and dotted curves correspond to half length of the pole equal to 0.5 (reference setting), 0.25 and 0.125 m respectively. It is surprising to see how the controller can handle the shorter pole easily and gracefully.

In the learning phase, we supply only two training data corresponding to initial conditions (10, 0) and (-10, 0) of the pole. Now it would be interesting to know how the FC (obtained from the reference setting) deals with other initial conditions. In this part, we monitor the pole behavior starting from other initial conditions which make the control goal even harder. Figure 14 shows

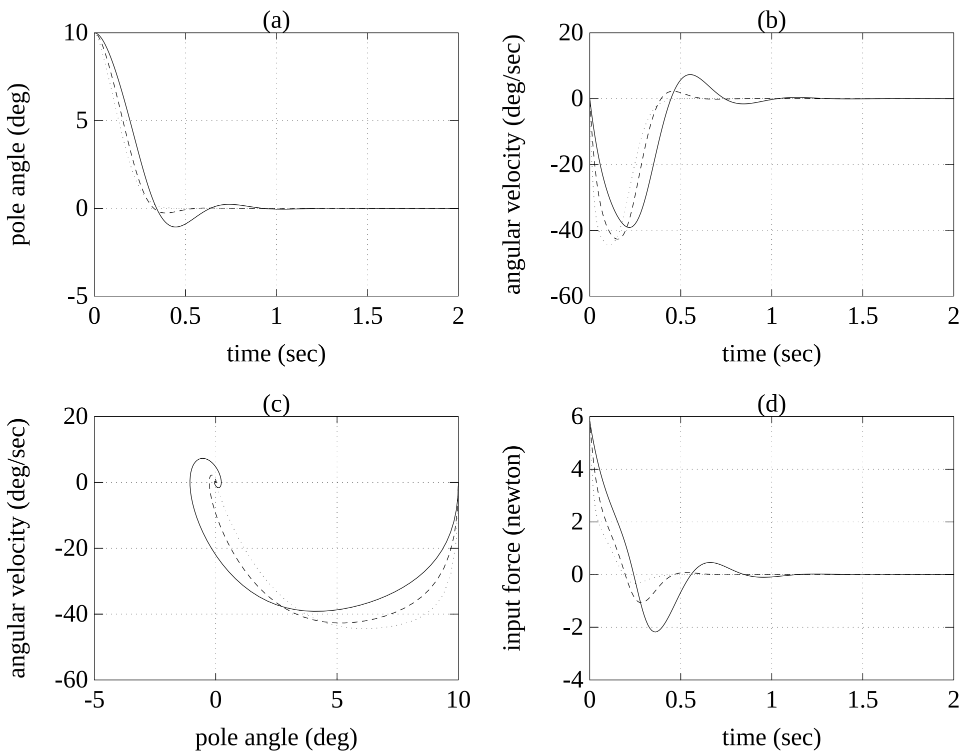

Figure 13: *(a) Pole angle; (b) pole angular velocity; (c) state space; (d) input force; (Solid, dashed and dotted curves correspond to half pole length of 0.5, 0.25 and 0.125 m, respectively.)*

the results, where the curve solid, dashed and dotted curves correspond to the initial conditions (10, 20), (15, 30) and (20, 40), respectively. Again, the same fuzzy controller can perform the control task starting from the unseen initial conditions. Figure 14 together with Figure 13 reveals the robustness and fault tolerance of the fuzzy controller obtained from the TBP.

V. Conclusions

We have proposed a generalized controller design methodology, called temporal back propagation (TBP), for constructing self-learning fuzzy controllers. This methodology employs the adaptive network as a building block and the back-propagation gradient method as the update procedure to minimize the difference between an actual trajectory and a given desired trajectory. Due to the flexibility of this methodology, we can easily customize it for a wide range of control applications. The inverted pendulum is used as a testbed to verify the effectiveness of the TBP and to exhibit the robustness and fault tolerance of the resulting fuzzy controller.

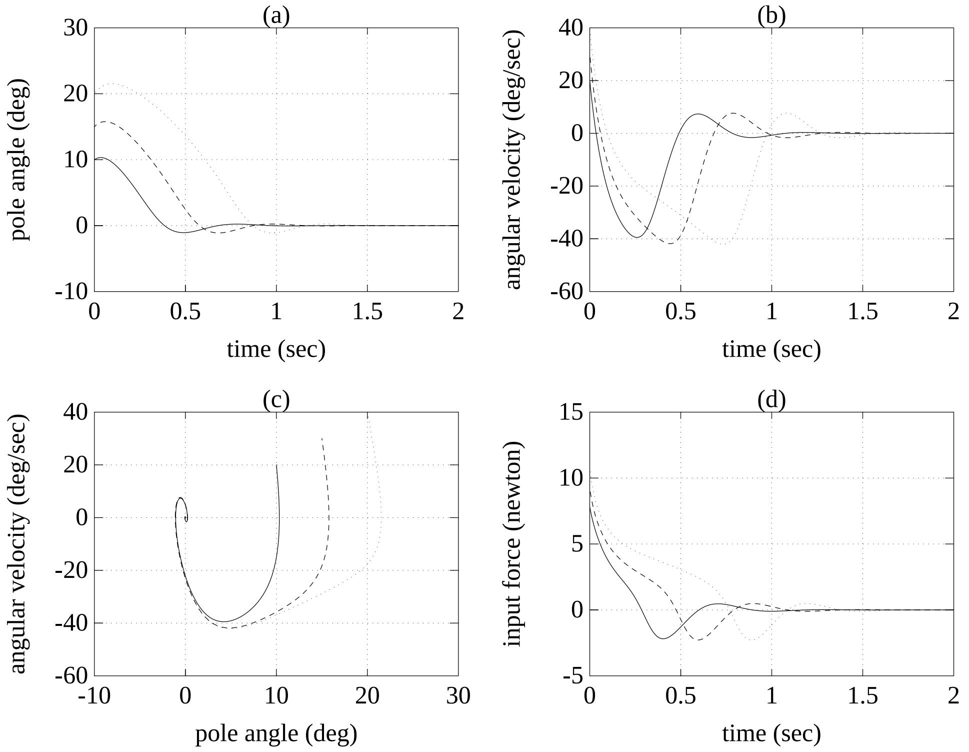

Figure 14: *(a) Pole angle; (b) pole angular velocity; (c) state space; (d) input force. (Solid, dashed and dotted curves correspond to initial conditions (10, 20), (15, 30) and (20, 40), respectively.)*

Best of all, the TBP is not tailored for fuzzy controllers only. Almost all non-pathogenic mathematical representations or equations (such as difference equations, ANFIS architecture, feed-forward neural networks, radial function basis networks, GMDH structure, etc) can be used to implement the plant block as well as the controller block. This provides us with plenty of freedom in choosing accurate and efficient models.

Acknowledgement

The guidance and help of Professor Lotfi A. Zadeh and other members of the "fuzzy group" at UC Berkeley is gratefully acknowledged.

Appendix

Each linguistic label used in the FC is characterized by 3 parameters as described in equation (10). If θ is in *degree*, $\dot{\theta}$ in *degree/sec*, the initial fuzzy if-then rules are

$$\begin{cases} If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0 \\ If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0 \end{cases} \tag{24}$$

where A_1, A_2, B_1 and B_2 are the linguistic labels characterized by (20, 2, −20), (20, 2, 20), (50, 2, −50) and (50, 2, 50), respectively. Figure 9 is the initial control action surface.

The final fuzzy if-then rules derived from the reference settings are

$$\begin{cases} If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0.0502 * \theta + 0.1646 * \dot{\theta} - 10.09 \\ If\ \theta\ is\ A_1\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0.0083 * \theta + 0.0119 * \dot{\theta} - 1.09 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_1,\ then\ force = 0.0083 * \theta + 0.0119 * \dot{\theta} + 1.09 \\ If\ \theta\ is\ A_2\ and\ \dot{\theta}\ is\ B_2,\ then\ force = 0.0502 * \theta + 0.1646 * \dot{\theta} + 10.09 \end{cases} \tag{25}$$

where A_1, A_2, B_1 and B_2 are the linguistic labels characterized by (−1.59, 2.34, −19.49), (−1.59, 2.34, 19.49), (85.51, 1.94, −23.21) and (85.51, 1.94, 23.21), respectively. Figure 10 is the final control action surface.

References

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

[2] H. R. Berenji. Refinement of approximate reasoning-based controllers by reinforcement learning. In *Proc. of the Eighth International Workshop of Machine Learning*, Evanston, Illinois, June 1991.

[3] R. H. Cannon. *Dynamics of Physical Systems.* McGraw-Hill, New York, 1967.

[4] W. T. Miller III, R. S. Sutton, and P. J. Werbos, editors. *Neural Networks for Control.* Massachusetts Institute of Technology, 1990.

[5] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 1(4):364–378, October 1971.

[6] J.-S. Roger Jang. Fuzzy modeling using generalized neural networks and Kalman filter algorithm. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 762–767, July 1991.

[7] J.-S. Roger Jang. Rule extraction using generalized neural networks. In *Proc. of the 4th IFSA World Congress*, July 1991.

[8] J.-S. Roger Jang. ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 1992. (Forthcoming).

[9] C.-C. Lee. Intelligent control based on fuzzy logic and neural network theory. In *Proc. of the Int. Conf. on Fuzzy Logic and Neural Networks, Iizuka*, pages 759–764, 1990.

[10] C.-C. Lee. A self-learning rule-based controller employing approximate reasoning and neural net concepts. *International Journal of Intelligent Systems*, 5(3):71–93, 1991.

[11] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[12] D. H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, pages 18–23, April 1990.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volum 1*, chapter 8, pages 318–362. The MIT Press, 1986.

[14] T. Takagi and M. Sugeno. Derivation of fuzzy control rules from human operator's control actions. *Proc. of the IFAC Symp. on Fuzzy Information, Knowledge Representation and Decision Analysis*, pages 55–60, July 1983.

[15] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.