Fuzzy Controller Design without Domain Experts

J.-S. Roger Jang
Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720

Abstract

In this paper, we present an innovative approach to the design of fuzzy controllers without resorting to the domain knowledge about the plants to be controlled. The fuzzy controller under consideration is implemented as an ANFIS (Adaptive-Network-based fuzzy inference system) while the plant can be implemented either as a neural network or as an ANFIS. Moreover, the plant can be implemented as an concise adaptive network if the difference or differential equations of the plant dynamics are available. Based on the learning rule of adaptive networks, the fuzzy controller can evolve automatically to acquire desired membership functions of the fuzzy if-then rules for achieving the control goal. An inverted pendulum system is used to test the proposed control scheme; the simulation results demonstrate its feasibility and robustness.

1 Introduction

Fuzzy controllers have recently found many successful applications in both industry and household appliances. For complex and/or ill-defined systems that are not subdued to conventional control techniques, fuzzy logic control provides a feasible alternative since it can capture the approximate, qualitative aspect of human reasoning. However, the performance of fuzzy controllers depends on two important factors, i.e., the soundness of knowledge acquisition techniques and the availability of human experts. These two factors severely restrict the application domains of fuzzy controllers.

We have proposed the architecture of ANFIS (Adaptive-Network-based Fuzzy Inference System) [5, 4, 3] which can bypass the first problem by eliciting the membership functions of fuzzy-if then rules directly from a desired input-output data set. The basics of adaptive networks and the ANFIS architecture are described in the next section.

This paper aims at the second problem: how to control a nonlinear systems through a self-learning mechanism that can derive the membership functions of the rules used by a fuzzy controller? In other words, without resorting to domain experts, we want to construct a fuzzy controller that can perform the control task of a regulator problem. This challenging problem has been investigated by other researchers such as Lee [6] and Berenji [1], primarily by reinforcement learning. Our approach is based on the adaptive network – a flexible building block that can be used to implement fuzzy controllers as well as the plants under consideration. The learning rule of adaptive networks can force the plant state to approach a desired state on a (time) step by (time) step basis.

We use the proposed approach to build a fuzzy controller for balancing an inverted pendulum system. It is shown that only 4 fuzzy if-then rules are necessary to perform the control task. Besides, the obtained controller is quite tolerant in dealing with initial conditions that deviate significantly from the origin.

*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.

0-7803-0236-2/92 $3.00 © 1992 IEEE

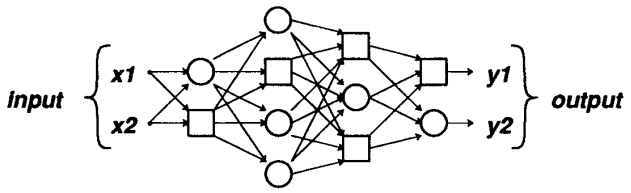

Figure 1: *An adaptive network.*

2 Basics of Adaptive Networks and ANFIS

In this section, we describe the basics of *adaptive network* and its learning rule. One of its special configurations, called *ANFIS* (*Adaptive-Network-based Fuzzy Inference System*), is also briefly introduced. For a more in-depth coverage, see [4, 3].

An adaptive network (Figure 1) is a multi-layer feedforward network in which each node performs a particular function (*node function*) on incoming signals as well as a set of parameters pertaining to this node. Usually we use both circle and square nodes in an adaptive network; a square node (adaptive node) has parameters while a circle node (fixed node) has none. In order to achieve a desired input-output mapping, these parameters are updated according to given training data and a gradient-based updating procedure described below.

Suppose that a given adaptive network has L layers and the k-th layer has $\#(k)$ nodes. Then the node function (or node output) at the i-th position of the k-th layer can be expressed as

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

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

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

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

where $T_{m,p}$ is the m-th component of p-th target output vector, $O_{m,p}^L$ is the m-th component of actual output vector produced by the presentation of the p-th input vector, and β_m ($¿$ 0)accounts for the relative importance of m-th output. Hence the overall error measure is $E = \sum_{p=1}^{P} E_p$.

The *error rate* for node output O is defined as the derivative of E_p with respect to O. For the output node at (L, i), the error rate can be calculated readily from equation (2):

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

For the internal node at (k, i), the error rate 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 rate of an internal node can be expressed as the linear combination of the error rates 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).

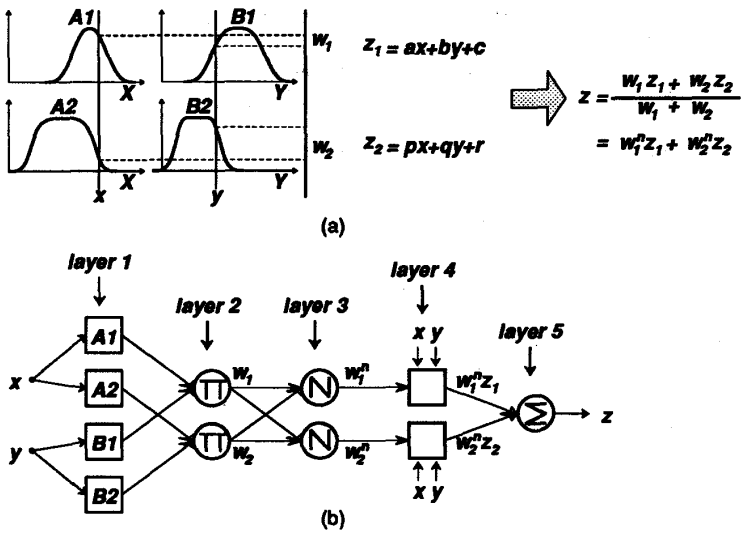

Figure 2: *(a) Two-input two-rule fuzzy inference system; (b) equivalent ANFIS.*

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

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

in which η is a updating rate of parameter α.

A two-input two-rule fuzzy inference system which utilizes fuzzy if-then rules of Takagi and Sugeno's type [7] is shown in Figure 2(a). Figure 2(b) is an equivalent ANFIS architecture in which node functions of the same layer have the same function type as described below:

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

$$O_i^1 = \mu_A(x) = \frac{1}{1 + [(\frac{x - c_i}{a_i})^2]^{b_i}}, \tag{8}$$

where x is one of the input variables, $\{a_i, b_i, c_i\}$ is the parameter set, and A is the linguistic label associated with this node function. Parameters in this layer will be 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 of a rule.

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. (See the equations in Figure 2(a).)

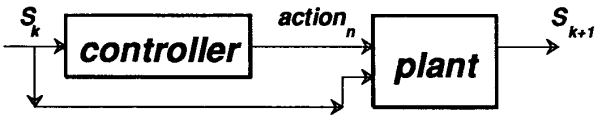

Figure 3: *Block diagram of a discrete control system.*

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

$$O_i^4 = w_n * (d_i * x + e_i * y + f_i), \tag{9}$$

where w_n is the output of layer 3, and $\{d_i, e_i, f_i\}$ is the parameter set. Parameters in this layer will be referred to as *consequent parameters*.

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

Functional speaking, the ANFIS architecture is completely equivalent to a fuzzy inference system (or fuzzy controller when used in control), as shown in Figure 2. However, by implementing a fuzzy controller as an ANFIS, we can easily employ the back-propagation-type learning procedure to find its parameters for achieving a minimal error measure.

3 Fuzzy Controller Design Based on Adaptive Networks

The block diagram of a feedback control system in discrete time domain is shown in Figure 3 where the delay through the controller is assumed to be small and the plant state variables are assumed to be accessible with accuracy. For a simple regulator problem, the control goal is to keep the plant state as close as possible to a desired zero state, or the origin.

Before finding a way to drive the plant state to a zero state in Figure 3, first we have to find a mathematical expressions for both the controller block and the plant block. This step is referred to as the *implementation* of the block diagram. Since the controller in question is a fuzzy controller, the obvious candidate for implementing the controller block is the ANFIS architecture introduced in the previous section. The reason is three-fold: (1) the functional behavior of the ANFIS is exact the same as a fuzzy controller with Takagi and Sugeno's fuzzy if-then rules [7]; (2) the learning rule of the ANFIS provides an effective method for achieving a desired input-output mapping; and (3) the ANFIS is actually an universal approximator provided the rule number is not restricted, which can be easily proved by Stone-Wierstrass theorem.

As for the implementation of the plant block, any mathematical expressions that can well imitate the input-output behavior of the plant are qualified candidates. From this point of view, the selection of plant implementation is quite flexible and it all depends on how well we know about the plant. For instance, if we know nothing about the plant except its input-output data pairs, then either artificial neural networks or the ANFIS is qualified candidate. On the other hand, if the plant dynamical equations are available with deterministic coefficients, then we can employ a simple adaptive network based on numerical methods (such as forward Euler integration, Runge-Kutta method, etc.) to implement the plant block in Figure 3. This part will become clear as we introduce the simulation in the next section.

No matter what kind of implementation we choose for the plant block, Figure 3 is still an adaptive network such that learning rule of adaptive networks can still be applied to find the desired input-output mapping from $\vec{S}_k$ to $\vec{S}_{k+1}$ for all k. Since the desired state in our case is a zero vector, the training data is of the form:

$$(\vec{S};\ \vec{0}), \tag{10}$$

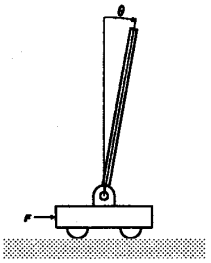

Figure 4: *The inverted pendulum system.*

where $\vec{S}$ is the input part and $\vec{0}$ (a zero vector) is the desired output after a small time step h. In other words, given any state $\vec{S}$ at time t, we expect the controller will drive the plant to a state at time $t+h$ as close as possible to a zero vector. Of course it is not realistic to expect the controller to get the job done in just one time step h; it is the collective effect of the training data that makes the plant state approach the origin. As a common trait to neural controller, it is hard, if not impossible, to prove the stability issue of this approach. However, the simulation results in the next section demonstrate its feasibility and robustness.

4 Application Example: Inverted Pendulum System

Figure 4 shows the schematic diagram of an inverted pendulum. Our control goal is to balance the pole by supplying appropriate force F to the cart. Let $x_1(t) = \theta(t)$ (angle of the pole with respect to the vertical axis) and $x_2(t) = \dot{\theta}(t)$ (angular velocity of the pole), then the state equations can be expressed as [2]:

$$\begin{cases} \dot{x_1} &= x_2, \\ \dot{x_2} &= \frac{g*sin(x_1)+cos(x_1)*(\frac{-F-m*l*x_2^2*sin(x_1)}{m_c+m})}{l*(\frac{4}{3}-\frac{m*cos^2(x_1)}{m_c+m})} \quad = \quad H_2(x_1, x_2, F), \end{cases} \tag{11}$$

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 $meter$, and F is the applied force in $newtons$.

Since the inverted pendulum is a well-behaved nonlinear dynamical system with precisely defined differential equations, we can use a two-step forward Euler integration to approximate its state at time $t+h$:

$$\begin{cases} x_1(t+0.5h) &= 0.5hx_2(t) + x_1(t), \\ x_2(t+0.5h) &= 0.5hH_2(x_1(t), x_2(t), F) + x_2(t), \end{cases} \tag{12}$$

$$\begin{cases} x_1(t+h) &= 0.5hx_2(t+0.5h) + x_1(t+0.5h), \\ x_2(t+h) &= 0.5hH_2(x_1(t+0.5h), x_2(t+0.5h), F) + x_2(t+0.5h), \end{cases} \tag{13}$$

where $h = 0.1$ sec. These four equations are the node functions of node 1, 2, 3 and 4, respectively, in the plant block of Figure 5. Note that F is a function of $x_1(t)$ and $x_2(t)$ and it is kept constant during the period $[t,\ t+h]$.

The controller block in Figure 5 is implemented as an ANFIS with four rules and each input is assigned two membership functions. In general, there is no simple way to determine in advance the minimal rule number required to achieve the control goal. And roughly speaking, the rule number is directly related to the degree of nonlinearity of the plant. In our simulation, the rule number is arbitrarily set to four and it turn out to be enough for achieving the control goal.

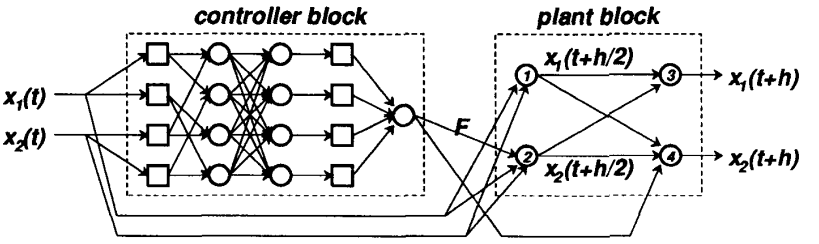

Figure 5: *Implementation of the block diagram.*

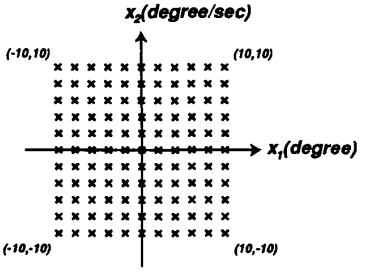

Figure 6: *Distribution of the input part of the training data set.*

The inverted pendulum is expected to start from an initial point nearby the origin in the state space, therefore the input parts of the training data are chosen to scatter evenly over a square region $[10, -10] \times [10, -10]$ in the state space, as shown in Figure 6. The total number of training data is 121.

We assume no prior knowledge (from a human operator's point of view) about the inverted pendulum system is available. Therefore consequent parameters are set to be all zeros (which means the control action is zero initially); the premise parameters are set in such a way that the membership functions can cover the operating range totally with sufficient overlapping with each other. It is observed that the variations of $\dot{\theta}$ are usually an order higher than those of θ, hence we set $\beta_1 = 10$ and $\beta_2 = 1$ in the error measure of equation (2).

The learning process converges pretty fast in less than 50 epochs. In fact, only after 10 epochs of learning, the fuzzy controller thus obtained can balance the pole already. The solid curve of Figure 8(a) exhibits how the pole angle and angular velocity converge to zero in about 2.5 *sec* from an initial condition (10, 10). The control action surface is shown as a 3-D plot in Figure 7. Furthermore, this controller can even balance the pole from some initial points that don't fall into the region of the training data set; the dashed curve in Figure 8(a) and (b) corresponds to the initial condition of (13, 15) and the dotted curve corresponds to (16, 20). Figure 8(c) and (d) demonstrate the trajectory and input force (controller's output) of each curve. (To reduce simulation error, Figure 8 is obtained through a fourth-order Runge-Kutta method where the time step equal to $h/10 = 0.01$ *sec.*)

5 Conclusions and Future Work

We have proposed a novel approach to the design of fuzzy controllers without resorting to domain knowledge of the plant under control. This approach employs the adaptive networks as building blocks and the back-propagation-type gradient method as a learning procedure to minimize the differences between the actual state and the desired state at each time step. Due to the flexibility of the proposed approach, we can easily customize it for a wide range of control applications. The

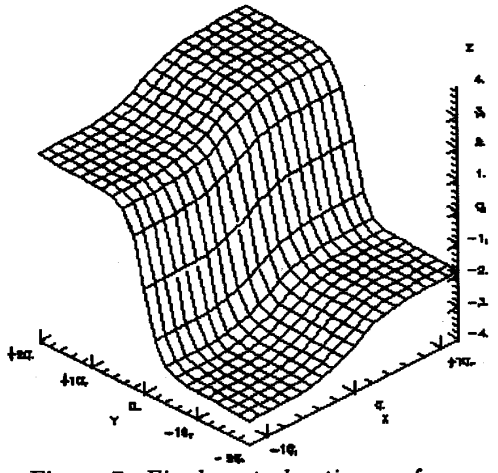

Figure 7: *Final control action surface.*

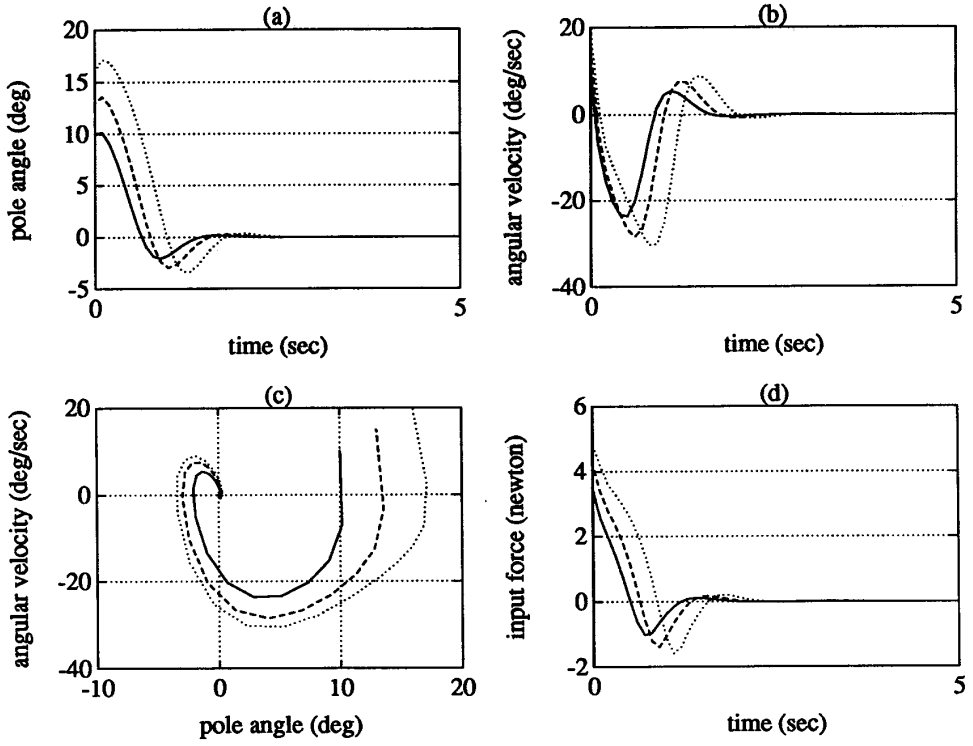

Figure 8: *(a) Pole angle; (b) pole angular velocity; (c) state-space trajectory; (d) input force. (The solid curves, dashed curves and dotted curves correspond to initial conditions (10, 10), (13, 15) and (16, 20), respectively.)*

inverted pendulum is used as a testbed to verify its effectiveness and to exhibit the robustness of the acquired fuzzy controller with respect to unseen initial conditions.

Most of all, the proposed control scheme is not tailored for fuzzy controller only. Almost all non-pathogenic mathematical representations or equations (such as difference equations, ANFIS, feedforward neural networks, radial function basis networks, GMDH structure, etc) can be used to build the the plant block as well as the controller block. This provide us with extreme freedom in choosing an accurate and efficient model.

This work is in its preliminary stage and there are a couple of promising directions worth further exploration. Some of the interesting future work includes:

- Self-learning in determining necessary number of fuzzy if-then rules.
- Automatic derivation of Lyapunov functions for verifying stability.
- Application to more challenging problems, such as tracking control.

References

[1] 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.

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

[3] J.-S. Jang. ANFIS: adaptive-network-based fuzzy inference systems. *IEEE Trans. on Systems, Man, and Cybernetics*, 1991. (submitted).

[4] J.-S. Jang. Fuzzy modeling using generalized neural networks and Kalman filter algorithm. In *Proc. of AAAI*, pages 762–767, July 1991.

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

[6] 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.

[7] 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.