

Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm

Jyh-Shing R. Jang

Department of Electrical Engineering and Computer Science
University of California, Berkeley, CA 94720

Abstract

We propose a new approach to build a fuzzy inference system of which the parameters can be updated to achieve a desired input-output mapping. The structure of the proposed fuzzy inference system is called *generalized neural networks*, and its learning procedure (rules to update parameters) is basically composed of a gradient descent algorithm and Kalman filter algorithm. Specifically, we first introduce the concept of generalized neural networks (GNN's) and develop a gradient-descent-based supervised learning procedure to update the GNN's parameters. Secondly, we observe that if the overall output of a GNN is a linear combination of some of its parameters, then these parameters can be identified by one-time application of Kalman filter algorithm to minimize the squared error. According to the simulation results, it is concluded that the proposed new fuzzy inference system can not only incorporate prior knowledge about the original system but also fine-tune the membership functions of the fuzzy rules as the training data set varies.

Introduction

It's known that conventional approaches to system modeling, which are based on mathematical tool (e.g., differential equations), perform poorly in dealing with complex and uncertain systems such as economical or ecological ones. Contrarily by employing *fuzzy if-then rules*, a *fuzzy inference system* can express the qualitative aspect of human reasoning without using any precise mathematical models of the system. This *fuzzy modeling* [Takagi and Sugeno, 1985, Sugeno and Kang, 1988] has found many practical applications in control, AI and OR fields, such as estimation, classification, inference, and prediction. However, some basic problems still plague this approach [Takagi and Hayashi, 1991]:

1. No formal ways to transform experiences or knowledge of human experts to the rule base and database of a fuzzy inference system.
2. The lack of adaptability or learning algorithms to tune the membership functions so as to minimize error measure.

The aim of this paper is to alleviate the above limitations by using a special structure called *GNN-based fuzzy inference system*. The concept of GNN (generalized neural network) and its gradient-descent-based learning procedure are introduced in the next section. We further use *Kalman filter algorithm* to speed up convergence and reduce the possibility of being trapped in local minima. The simulation results to verify the proposed approach are demonstrated through 3-dimension diagrams. To sum up, some general comments and discussions are given in the last section.

Generalized Neural Networks

A generalized neural network (GNN) (Figure 1) is a multi-layer feed-forward 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. If the set of parameters is empty, then we use a circle to denote this node, otherwise a square. The exact forms of node functions may vary from node to node, and the choice of node functions depends on the overall function that a GNN is designed to carry out. (Note that the links in a GNN only serve the transmission of signals between nodes; no weight is associated with each link.)

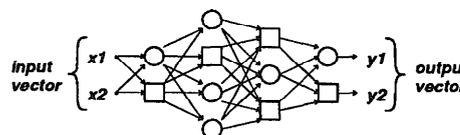


Figure 1: A generalized neural network (GNN).

The parameter set of a GNN is the union of the parameter set of each node. In order to minimize the

*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.

output error measure of a GNN, these parameters are updated according to given training data and a learning algorithm described later.

Cybenko [Cybenko, 1989] showed that a continuous neural network (NN) with two hidden layers and any fixed continuous sigmoidal nonlinear can approximate any continuous function arbitrarily well on a compact set. Therefore a GNN node can always be replaced by an ordinary NN with the same input-output characteristics. In this context, GNN can be looked upon as a super set of NN.

Suppose a given GNN 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

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

where a, b, c , etc. are parameters pertaining to this node.

Assuming the given training data set has P entries, then we can define the *error measure* (or *energy function*) on p -th ($1 \leq p \leq P$) entry of training data as the square of error:

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

where $T_{m,p}$ is the m -th component of p -th target output vector, and O_m^L is the m -th component of actual output vector produced by the presentation of the p -th input vector. (For brevity, we omit the subscript p in O_m^L .) Hence the overall error measure is $E = \sum_{p=1}^P E_p$. Now we want to develop a learning procedure that implements gradient descent in E over the parameter space.

For the output-layer node at (L, i) , we can calculate $\frac{\partial E_p}{\partial O_i^L}$ readily from Equation 2:

$$\frac{\partial E_p}{\partial O_i^L} = -2(T_{i,p} - O_i^L) \quad (3)$$

For the internal node at (k, i) , we can use chain rule to write $\frac{\partial E_p}{\partial O_i^k}$ as

$$\frac{\partial E_p}{\partial O_i^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_m^{k+1}} \frac{\partial O_m^{k+1}}{\partial O_i^k} \quad (4)$$

where $1 \leq k \leq L-1$. Therefore for all $1 \leq k \leq L$ and $1 \leq i \leq \#(k)$, we can find $\frac{\partial E_p}{\partial O_i^k}$ by Equation 3 and 4. Now if α is a parameter of the given GNN, we have

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \frac{\partial O^*}{\partial \alpha} \quad (5)$$

where S is the set of nodes whose outputs depend on α . The derivative of overall error measure E to α is then calculated as:

$$\frac{\partial E}{\partial \alpha} = \sum_{p=1}^P \frac{\partial E_p}{\partial \alpha} \quad (6)$$

Accordingly, the update amount for the generic parameter α is

$$\Delta \alpha = -\eta \frac{\partial E}{\partial \alpha} \quad (7)$$

where η is a learning rate and it can be further expressed as

$$\eta = \frac{k}{\sqrt{\sum_{\alpha} (\frac{\partial E}{\partial \alpha})^2}} \quad (8)$$

where k is the step size of the gradient descent.

GNN-based Fuzzy Inference System

An example of fuzzy if-then rules used in a fuzzy inference system is

If pressure is high and temperature is low, then volume is small.

where *pressure* and *temperature* are input variables, *volume* is an output variable, *high*, *low* and *small* are linguistic terms [Zadeh, 1988, Zadeh, 1989] characterized by appropriate *membership functions* [Zadeh, 1965]. Each fuzzy rule represents a local description of the system's behavior.

Several types of reasoning methods [Lee, 1990a, Lee, 1990b] used in fuzzy models have been proposed during the past years. Here we adopt the one proposed by Sugeno [Takagi and Sugeno, 1983], see Figure 2. Note that the firing strength (w_1 and w_2 in Figure 2), or weight, of each fuzzy rule is calculated as the product of the membership values in the premise part, and the final output is obtained as the weighted average of each rule's consequence.

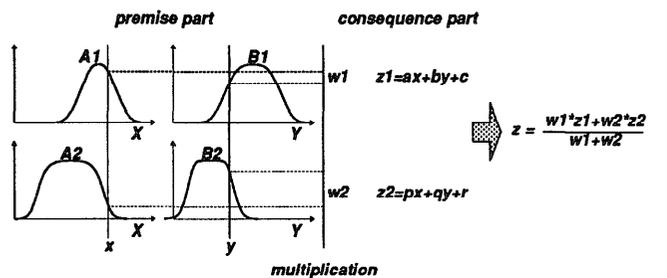


Figure 2: The reasoning mechanism adopted

For illustration, we assume the system to be modeled has (1) two input variables x and y , each of which has three membership functions associated with it, (2) and one output variable z . Ideally, these three membership functions correspond to three commonly used

linguistic terms, i.e., "small", "medium" and "large", to describe the input variables. The basic configuration of the GNN-based fuzzy inference system is shown in Figure 3, where nodes in the same layer have the same type of node functions explained below.

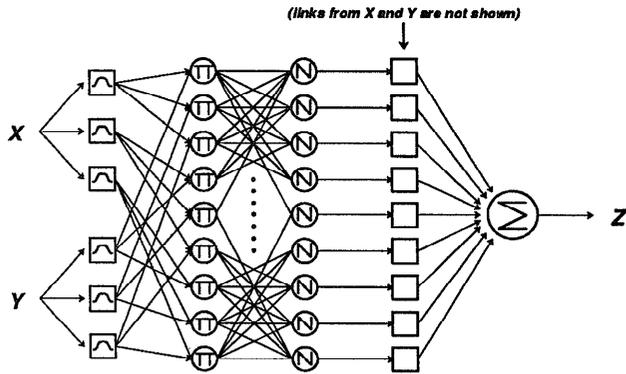


Figure 3: A GNN-based fuzzy inference system.

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

$$O^1 = \mu_A(x) = \frac{1}{1 + \left[\left(\frac{x-c}{a}\right)^2\right]^b} \quad (9)$$

where x is one of the input variables, $\{a, b, c\}$ is the parameter set, and A is the linguistic term. As the values of a , b and c change, this bell-shaped node function varies accordingly, thus exhibiting various concepts of corresponding linguistic term. Parameters in this layer are usually called *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 corresponds to 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.

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

$$O^4 = w_n * (d * x + e * y + f) \quad (10)$$

where w_n are from layer 3, and $\{d, e, f\}$ is the parameter set. Parameters in this layer are usually called *consequence parameters*.

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

Thus we have constructed a fuzzy inference system by a GNN with 5 layers, 34 nodes, and 45 parameters (18 in layer 1, 27 in layer 4). Then the proposed learning procedure can be applied to tune the parameters according to given training data. One way to speed up the tuning process is to employ Kalman filter algorithm which is to be discussed in the next section.

Kalman Filter Algorithm

From Figure 3, it is observed that given the values of the premise parameters and P entries of training data, we can form P linear equations in terms of the consequence parameters. For simplicity, assume there are m consequence parameters and the resulting linear equations can be expressed in the following matrix form:

$$AX = B \quad (11)$$

where the elements of X are consequence parameters. Several approaches have been developed to solve this kind of over-constrained simultaneous linear equations, and one of the most concise is

$$X^* = (A^T A)^{-1} A^T B \quad (12)$$

where A^T is the transpose of A , and $(A^T A)^{-1} A^T$ is called the *pseudo-inverse* of A if $A^T A$ is non-singular.

In many cases, the row vectors of matrix A (and corresponding elements in B) are obtained sequentially, hence it may be desirable to compute the least-square estimate of X in Equation 11 recursively. Let the i th row vector of matrix A defined in Equation 11 be a_i and the i th element of B be b_i , then X can be calculated recursively using the following formulas [Ikeda *et al.*, 1976, Astrom and Wittenmark, 1984]:

$$\left. \begin{aligned} X_{i+1} &= X_i + S_{i+1} a_{i+1}^T (b_{i+1} - a_{i+1} x_i) \\ S_{i+1} &= S_i - \frac{S_i a_{i+1}^T a_{i+1} S_i}{1 + a_{i+1}^T S_i a_{i+1}}, \quad i = 0, 1, \dots, P-1 \\ X &= X_P \end{aligned} \right\} \quad (13)$$

with initial conditions

$$X_0 = 0 \quad \text{and} \quad S_0 = \gamma I. \quad (14)$$

where γ is a positive big number, I is the identity matrix of dimension $m \times m$.

The least-square estimate of X can be interpreted as a Kalman filter [Kalman, 1960] for the process

$$X(k+1) = X(k) \quad (15)$$

$$Y(k) = A(k)X(k) + \text{noise} \quad (16)$$

where $X(k) = X_k$, $Y(k) = b_k$ and $A(k) = a_k$. Therefore the formulas in Equation 13 are usually referred to as *Kalman filter algorithm*.

Simulation Results

In the first example, the training data are obtained from a fuzzy inference system with 2 inputs and 1 output, where 3 membership functions are assigned to each input variable. We use 121 training data which are sampled uniformly from $[-10, 10] \times [-10, 10]$ of the input space of the original system.

The GNN used here is exactly the same as Figure 3. The initial values of the premise parameters are set in such a way that the membership functions along X and Y axis satisfy ϵ completeness [Lee, 1990a, Lee, 1990b] ($\epsilon = 0.5$ in our case), *normality*

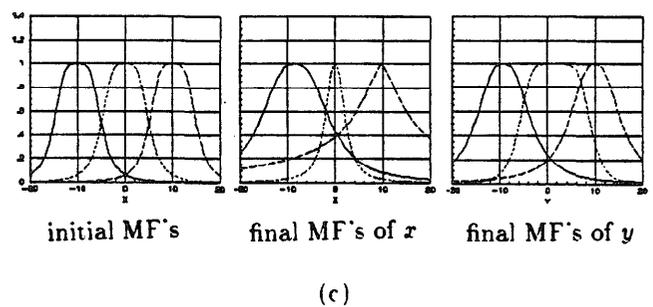
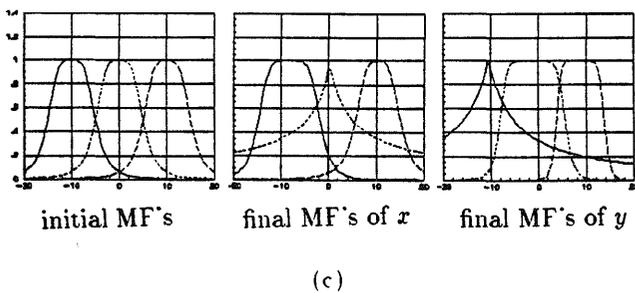
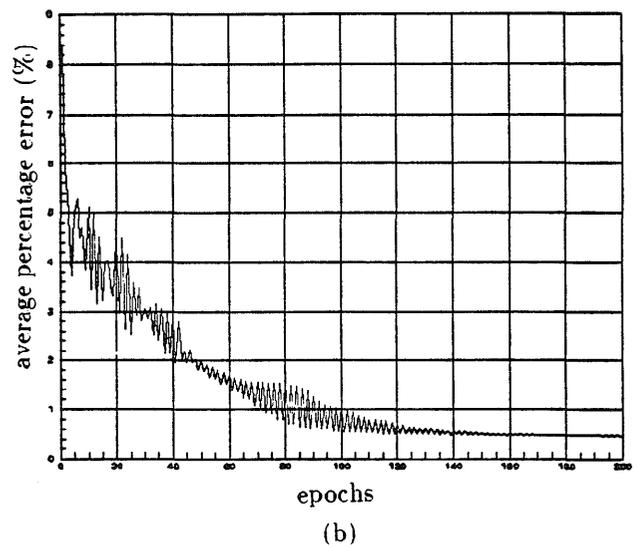
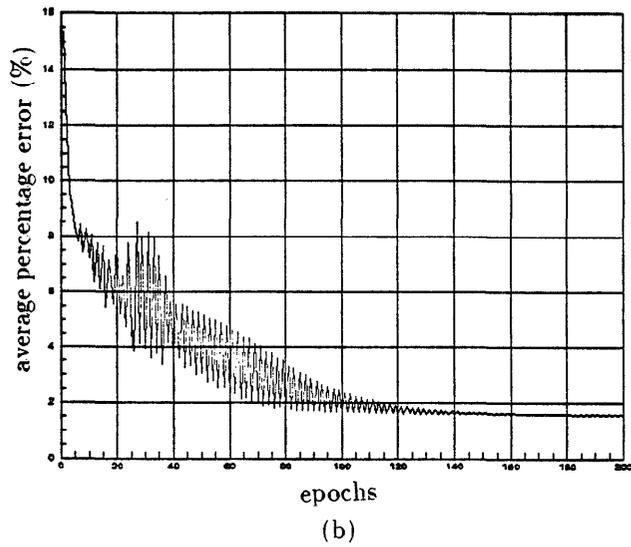
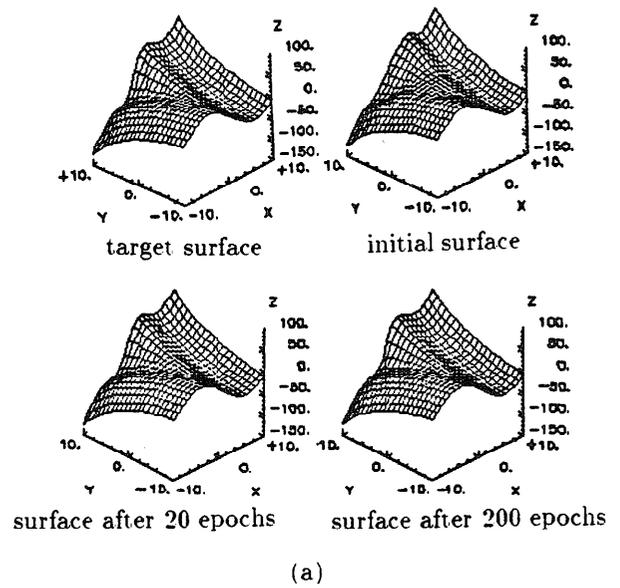
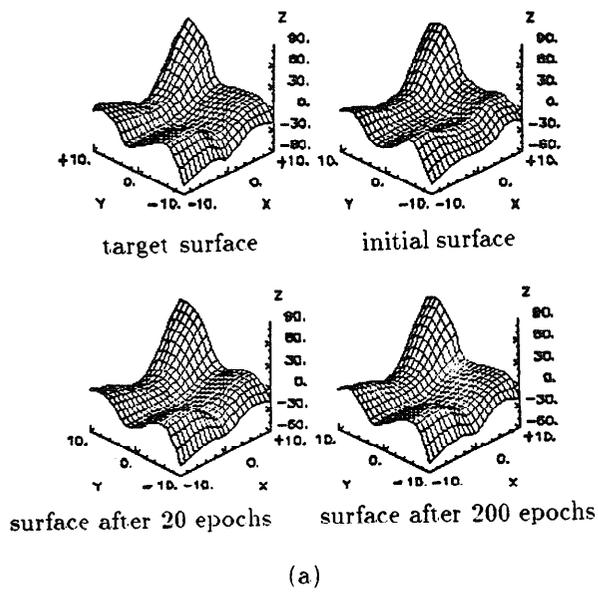
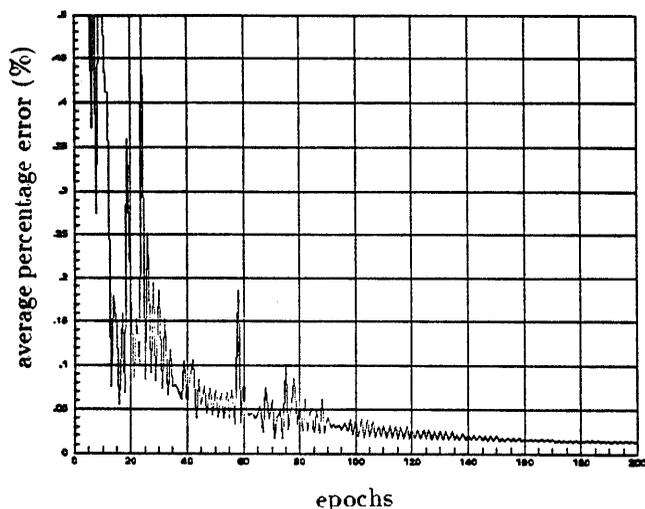
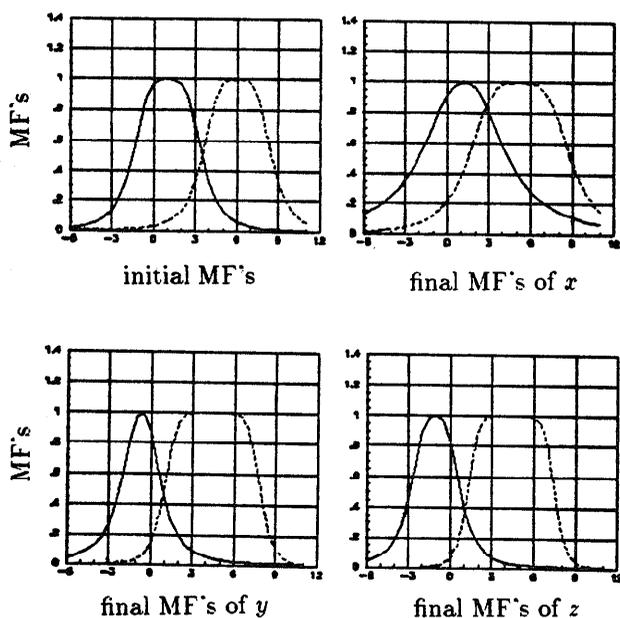


Figure 4: Example 1, (a) target surface and identified surfaces in different stages, (b) average percentage error, (c) initial and final membership functions (MF's).

Figure 5: Example 2, (a) target surface and identified surfaces in different stages, (b) average percentage error, (c) initial and final membership functions (MF's).



(a)



(b)

Figure 6: Example 3, (a) average percentage error, (b) initial and final membership functions (MF's).

and *convexity* [Kaufmann and Gupta, 1985]. Though these initial membership functions are set heuristically and subjectively, they do provide an easy interpretation parallel to human thinking. The initial and final membership functions are shown in Figure 4(c). The consequence parameters can always be found by Kalman filter algorithm as long as the premise parameters are fixed. Therefore, the initial values of consequence parameters are irrelevant here. The 3-D diagram of the training data is shown as the target surface in Figure 4(a). Other identified surfaces at different epoch numbers are also shown in that figure. (Note that the "initial surface" is obtained after the first time the consequence parameters have been identified, so it looks similar to the target surface already.) In order to evaluate the performance of the GNN-based fuzzy inference system, we define *average percentage error* (APE) as

$$APE = \frac{\sum_{i=1}^P |T(i) - O(i)|}{\sum_{i=1}^P |T(i)|} * 100\% \quad (17)$$

where P is the number of training data ($P = 121$ in this example); $T(i)$ and $O(i)$ are i -th desired output and calculated output, respectively. Though the final values of parameters are not the same as those used in the original fuzzy inference system, the final surface after 200 epochs is close to the target surface with APE equal to 1.5674%.

In the second example, all the settings are the same as those in the previous example except that the training data are obtained from a nonlinear function (see the target function in Figure 5(a))

$$z = (3e^{\frac{x}{10}} - 1) * 15 * \tanh\left(\frac{x}{2}\right) + (4 + e^{\frac{y}{10}}) * 8 * \sin\left(\frac{(x+4)\pi}{10}\right) \quad (18)$$

The simulation results are shown in Figure 5. After 200 epochs, we end up with an APE equal to 0.4641%, which is quite good considering the complexity of the target function.

In the third example, the training data are obtained from

$$output = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2 \quad (19)$$

This example has been used by Takagi [Takagi and Hayashi, 1991], Sugeno [Sugeno and Kang, 1988] and Kondo [Kondo, 1986] to verify their approaches. The GNN used here has 18 premise parameters in layer 1 and 24 consequence parameters in layer 4. We use 216 training data which are sampled uniformly from $[1, 6] \times [1, 6] \times [1, 6]$. In order to compare the simulation results with previously reported research, we use a different APE:

$$APE = \sum_{i=1}^P \frac{|T(i) - O(i)|}{|T(i)|} * 100\% \quad (20)$$

(We cannot use this definition in the previous examples since the denominator could be zero.) The

simulation results are shown in Figure 6. (The target and identified surfaces are hyper-surfaces in 4-D space, so we cannot display them as before.) After 200 epochs, the APE is driven to 0.01357 %, which is much better than those (0.59 % and 4.7 %) reported by [Sugeno and Kang, 1988] and [Kondo, 1986].

Comments and Discussion

In this paper, we have successfully solved the second problem mentioned in Section 1. The first problem, however, is considered partially solved since in our approach the number of membership functions is pre-specified, as well as the number of rules. A promising future work is to use clustering method to roughly find the number of membership functions needed, then apply our approach to find their optimal shapes.

Since the proposed learning algorithm is a gradient descent procedure, sometimes it could get stuck around local minima. However, the final average percentage error is still acceptable even though the global minimum has not been found, as shown in the second example in Section 5. Furthermore in a GNN-based fuzzy inference system, if prior knowledge of the original system is available, we can incorporate it into the initial values of the parameters. This provides an initial point close to the optimal one in the (premise) parameter space, thus decreasing the possibility of being trapped in local minima during the learning process.

References

- Astrom, K. J. and Wittenmark, B. 1984. *Computer controller systems: theory and design*. Prentice-Hall, Inc.
- Cybenko, G. 1989. Continuous value neural networks with two hidden layers are sufficient. *Math Contr. Signal and Sys.* 2:303-314.
- Ikeda, S.; Ochiai, M.; and Sawaragi, Y. 1976. Sequential GMDH algorithm and its application to river flow prediction. *IEEE Trans. on Systems, Man, and Cybernetics* 6(7):473-479.
- Kalman, R. E. 1960. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering* 35-45.
- Kaufmann, Arnold and Gupta, Madan M. 1985. *Introduction to Fuzzy Arithmetic*. Van Nostrand Reinhold Company.
- Kondo, T. 1986. Revised GMDH algorithm estimating degree of the complete polynomial. *Tran. of the Society of Instrument and Control Engineers* 22(9):928-934. (Japanese).
- Lee, Chuen-Chien 1990a. fuzzy logic in control systems: fuzzy logic controller-part 1. *IEEE Trans. on Systems, Man, and Cybernetics* 20(2):404-418.
- Lee, Chuen-Chien 1990b. fuzzy logic in control systems: fuzzy logic controller-part 2. *IEEE Trans. on Systems, Man, and Cybernetics* 20(2):419-435.

Sugeno, M. and Kang, G. T. 1988. Structure identification of fuzzy model. *Fuzzy Sets and Systems* 28:15-33.

Takagi, Hideyuke and Hayashi, Isao 1991. Artificial-neural-network-driven fuzzy reasoning. *International Journal of Approximate Reasoning*. To appear.

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

Takagi, T. and Sugeno, M. 1985. Fuzzy identification of systems and its applications to modeling and control. *IEEE Trans. on Systems, Man, and Cybernetics* 15:116-132.

Zadeh, Lotfi A. 1965. Fuzzy sets. *Information and Control* 8:338-353.

Zadeh, Lotfi A. 1988. Fuzzy logic. *Computer* 1(4):83-93.

Zadeh, Lotfi A. 1989. Knowledge representation in fuzzy logic. *IEEE Trans. on Knowledge and Data Engineering* 1:89-100.