Matrix Formulation of Fuzzy Rule-Based Systems

A. Lotfi, H. C. Andersen, and A. C. Tsoi

Abstract— **In this paper, a matrix formulation of fuzzy rule-based systems is introduced. A gradient descent training algorithm for the determination of the unknown parameters can also be expressed in a matrix form for various adaptive fuzzy networks. When converting a rule-based system to the proposed matrix formulation, only three sets of linear/nonlinear equations are required instead of set of rules and an inference mechanism. There are a number of advantages which the matrix formulation has compared with the linguistic approach. Firstly, it obviates the differences among the various architectures; and secondly, it is much easier to organize data in the implementation or simulation of the fuzzy system. The formulation will be illustrated by a number of examples.**

I. Introduction

There has been much research activity in fuzzy rule-based systems concerning the conceptual understanding of linguistic variables and fuzzy values. There are numerous proposals for inference mechanisms [18], [19], rule extractions [13], [26], and membership function assignments [1], [9], [12], [15], [20], [22].

For a given input-output mapping, using a fuzzy rule based system, it is possible to adjust the following parameters: 1) the membership function classes, 2) the number of rules used, and 3) the inference mechanism, to achieve an approximation. It was shown in [14] that the effects of changing the membership functions are dominant over the other two factors. Consequently, much research has been directed to the problem of membership function assignment in fuzzy neural networks [8], [12], adaptive network based fuzzy inference systems [10], and other adaptive structures for fuzzy models [15].

We propose a matrix formulation for fuzzy rule-based systems which gives a unified treatment of different already proposed fuzzy if-then rules and fuzzy inferences. This matrix formulation makes training algorithms very simple and understandable.

Since the first successful application of a fuzzy rule-based system as a controller of a simple plant by Mamdani and Assilian [16], it was always considered that the fuzzy rule-based controller gives an alternate paradigm to the "analytic" control theory. The traditional control theory is analytic in that it is based on methods relating to differential or difference equations, e.g., Laplace transform. Furthermore, it is often model based, i.e., it assumes a model of the plant in the analysis and design of a controller [11]. In contradistinction to this, the fuzzy control approach is based on the decision making concept in artificial intelligence [17]. Although there has been wide-spread acceptance of this "nonanalytic" approach for controller design, the analytic and nonanalytic approaches have grown independently.

Since most of modern control systems can be modeled using a set of first order matrix differential equations—the so-called state space approach—it would be useful to attempt a formulation of the fuzzy control systems using a similar approach. However, a major stumbling block in this endeavor is that the fuzzy control system is one with a multiplicative nonlinearity. This means that, although we are able to write the system in a matrix form, it is quite complicated, and hence would make further analysis very difficult.

We propose to overcome this stumbling block by introducing an approximation using a single-layer neural network to the multiplicative nonlinearity. It is shown that this approximation is valid over a range of input values. By using this approximation, the fuzzy neural network paradigm can be represented using matrix terminology.

With our proposed matrix model, a fuzzy rule-based system can be decomposed into a three layer feedforward neural network, each layer is represented by a set of linear or nonlinear difference equations. In the first layer, the membership function parameters of the antecedent are given. The second layer is an algorithm for the approximation of the multiplication or minimum connective operator *and*. The third layer operates as consequent parameters and the defuzzification method. Note that this connectionist architecture is not fully connected, i.e., each neuron may have a local receptive field [3], rather than connections to *all* the neurons in the previous layer.

Once the fuzzy rule-based system is represented in matrix terminology, it is simple to derive a gradient descent training algorithm to estimate the unknown parameters using an error function. We propose again to express this training algorithm in terms of matrices, thus finishing a "transparent" view,[1] and allowing for its simple implementation.

The organization of the rest of the paper is as follows: different fuzzy rule-based systems and related inference mechanisms will be briefly described in Section II, followed by a matrix formulation of a multi-layer perceptron in Section III to set the background for the proposed method. A neural network structure of the proposed approximation of the multiplicative operator, and subsequent explanation for different layers of the fuzzy rule-based system will be introduced in Section IV. The matrix formulation of a fuzzy rule-based system is given in Section V, and in Section VI gradient descent training algorithms will be explained. Some numerical examples are presented in Section VII and some conclusions will be drawn in Section VIII.

II. Fuzzy Rule-Based Systems

In this paper, we will present a matrix formulation of Fuzzy Inference Systems (FIS's). It has been shown that FIS's can be represented by three layer feedforward neural networks and it is on this structure that we base our matrix formulation for FIS. Firstly, we will review the structure of FIS and some specific defuzzification algorithms, then we will show how to represent the system by three sets of matrix equations.

First, consider a fuzzy rule base containing n rules, R^i, $i = 1, 2, \ldots, n$ with p inputs, x_j, $j = 1, 2, \cdots, p$, and q outputs, y_ℓ, $\ell = 1, 2, \cdots, q$ as follows:

$$\begin{array}{l} R^i\text{: If } x \text{ is } \tilde{\mathcal{M}}^i \text{ then } y \text{ is } \tilde{\mathcal{N}}^i, \text{ else} \\ O\text{: } x \text{ is } \mathcal{M}' \\ \hline \triangle\ y \text{ is } \mathcal{N}' \end{array}$$

The antecedent vector $x^T = [x_1, x_2, \cdots, x_j, \cdots, x_p]$ is a p vector with elements which are linguistic variables in the universe of $U^T = [U_1, U_2, \cdots, U_p]$. The superscript T represents the transpose of a vector. The consequent vector $y^T = [y_1, y_2, \cdots, y_\ell, \cdots, y_q]$ is a q vector whose elements are linguistic variables in the universe of $V^T = [V_1, V_2, \cdots, V_q]$. We further assume that the universe of antecedent and consequent, i.e., U and V, are limited to a specific domain interval as indicated below:

$$\begin{aligned} U_j &= [U_j^- \quad U_j^+] \quad j = 1, \cdots, p \\ V_\ell &= [V_\ell^- \quad V_\ell^+] \quad \ell = 1, \cdots, q. \end{aligned} \tag{1}$$

Manuscript received April 24, 1994; revised December 28, 1994.

The authors are with the Department of Electrical and Computer Engineering, University of Queensland, Brisbane 4072, Australia (lotfia@elec.uq.oz.au; andersen @elec.uq.oz.au; act@elec.uq.oz.au).

Publisher Item Identifier S 1083-4419(96)02298-4.

[1] "Transparency"is used to denote that one may clearly see the relationship among the variables in obtaining the gradient descent training algorithm. It is used in contrast to the more traditional indexed type of notation, which tends to make the relationships more "opaque" to observe.

Vector $\tilde{\mathcal{M}}^i = [\tilde{M}_1^i, \tilde{M}_2^i, \cdots, \tilde{M}_j^i, \cdots, \tilde{M}_p^i]$ is a vector of linguistic values and $\tilde{\mathcal{N}}^i = [\tilde{N}_1^i, \tilde{N}_2^i, \cdots, \tilde{N}_\ell^i, \cdots, \tilde{N}_q^i]$ is a vector of fuzzy values referring to the fuzzy variables x and y, respectively. The notation $\sim$ represents the fuzzy concept in contrast with crisp values. Vector $\mathcal{M}'$ and $\mathcal{N}' = [N_1', N_2', \cdots, N_\ell', \cdots, N_q']$ are, respectively, the crisp observation and conclusion vectors. The number of individual membership functions for a specific input value x_j $(\tilde{M}_j^1, \tilde{M}_j^2, \cdots, \tilde{M}_j^n)$ is $\hat{p}_j$. The number of individual membership functions for a specific output variable y_ℓ $(\tilde{N}_\ell^1, \tilde{N}_\ell^2, \cdots, \tilde{N}_\ell^n)$ is $\hat{q}_\ell$. Note that $\hat{p}_j \leq n$ and $\hat{q}_\ell \leq n$.

To make an inference "Y is $\mathcal{N}'$" from a set of rules R and an observation O, several types of fuzzy reasoning mechanisms have been proposed. There are different fuzzy reasoning mechanisms for various types of fuzzy if-then rules. We consider three different forms of fuzzy if-then rules and their related reasoning mechanisms for deducing the vector of crisp decision $\mathcal{N}'$ which has been used widely in the literature [21], [23], [27].

Type I: The first form of fuzzy if-then rules is the simplest, but is relatively effective in control applications. The corresponding consequent of rules are crisp numbers rather than linguistic values, i.e., $\tilde{\mathcal{N}}^i = \mathcal{N}^i$. The number of individual consequent parameters $\hat{q}_\ell$ is equal to n. The crisp decision N_ℓ' is calculated as:

$$N_\ell' = \frac{\sum_{i=1}^{n} w_i N_\ell^i}{\sum_{i=1}^{n} w_i}, \quad \ell = 1, 2, \cdots, q \tag{2}$$

where w_i is the rule firing strength given by:

$$w_i = \prod_{j=1}^{p} \tilde{M}_j^i(x_j). \tag{3}$$

Wang and Mendel [27] have proven that the fuzzy representation in (2) is a universal approximator. This model will be referred as the *semi-fuzzy if-then rule* model. It can be thought of as a simplification of Type II which is introduced below.

Type II: Another type of fuzzy if-then rule was proposed by Takagi and Sugeno [23]. In this method, the fuzzy values are involved only in the antecedent part of premises and the consequent is a linear function of its inputs i.e., $N_\ell^i + \sum_{j=1}^{p} (\hat{N}_j^i)_\ell x_j$. The output crisp decision N_ℓ' is obtained as:

$$N_\ell' = \sum_{i=1}^{n} \overline{w}_i f_{i\ell} \quad \ell = 1, 2, \cdots, q \tag{4}$$

where

$$\overline{w}_i = \frac{w_i}{\sum_{i=1}^{n} w_i}$$

$$f_{i\ell} = N_\ell^i + \sum_{j=1}^{p} (\hat{N}_j^i)_\ell x_j \tag{5}$$

The number of individual consequent parameters $\hat{q}_\ell = n$. $(\hat{N}_j^i)_\ell$ and N_ℓ^i are $n \times q + p(n \times q)$ constant parameters which must be determined.

Type III: The third fuzzy if-then rule is a general model with linguistic fuzzy values in both the antecedent and consequent parts of rules.

Pacini and Kosko [21] have shown that, if the correlation product inference determines the output fuzzy values, the global centroid N_ℓ' can be computed from local consequent premise centroids. Therefore, the output decision is given by:

$$N_\ell' = \frac{\sum_{i=1}^{n} w_i N_\ell^i S_\ell^i}{\sum_{i=1}^{n} w_i S_\ell^i} \quad \ell = 1, \cdots, q \tag{6}$$

where w_i, N_ℓ^i, and S_ℓ^i are the rules' firing strength (as defined earlier), local centroid, and area of consequent premises, respectively. When the number of consequent individual membership functions, $\hat{q}_\ell$, is less than number of rules, n the (6) can be rewritten as follows [2]:

$$N_\ell' = \frac{\sum_{r=1}^{\hat{q}_\ell} \Omega_r N_\ell^r S_\ell^r}{\sum_{r=1}^{\hat{q}_\ell} \Omega_r S_\ell^r} \quad \ell = 1, \cdots, q \tag{7}$$

where

$$\Omega_r = \sum w_i \quad \forall i \in 1, 2, \cdots, n \quad r = 1, \cdots, \hat{q}_\ell. \tag{8}$$

III. Matrix Formulation of a Single Layer Perceptron

Before describing the matrix formulation of fuzzy rule-based systems, it would be beneficial to clarify the concept of matrix formulation of a single-layer perceptron (SLP). This formulation was first proposed in [25] as a method for formulating a multilayer perceptron (MLP) using matrix representations. It has been used successfully for the formulation of an extensive class of recurrent networks [24].

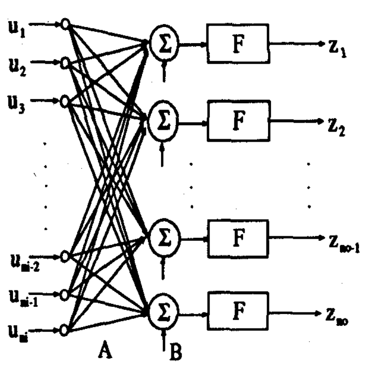


Fig. 1. One layer neural network with n_i inputs and n_o outputs.

Consider a one layer network with n_i inputs and n_o outputs. Each node function can be considered a nonlinear operation on a linear combination of inputs (see Fig. 1). We can express this functionality in matrix form as follows:

$$z = F(Au + B) \tag{9}$$

where $z^T = [z_1 \quad z_2 \quad \cdots \quad z_{n_o}]$ and $u^T = [u_1 \quad u_2 \quad \cdots \quad u_{n_i}]$ are the output and input vectors, respectively. A and B are matrices, of dimensions $n_o \times n_i$ and $n_o \times 1$, respectively, which represent the synaptic weights connecting the input nodes and the hidden layer neurons, and the threshold weights. $F^T = [f(.) \quad f(.) \quad \cdots \quad f(.)]$ is a n_o vector function representing the nonlinear activation functions. This nonlinear activation function $f(.)$ can be a sigmoid, a hyperbolic tangent, or any other similar function [7]. It is interesting to note that this formulation can be extended to a network containing more than one layers.

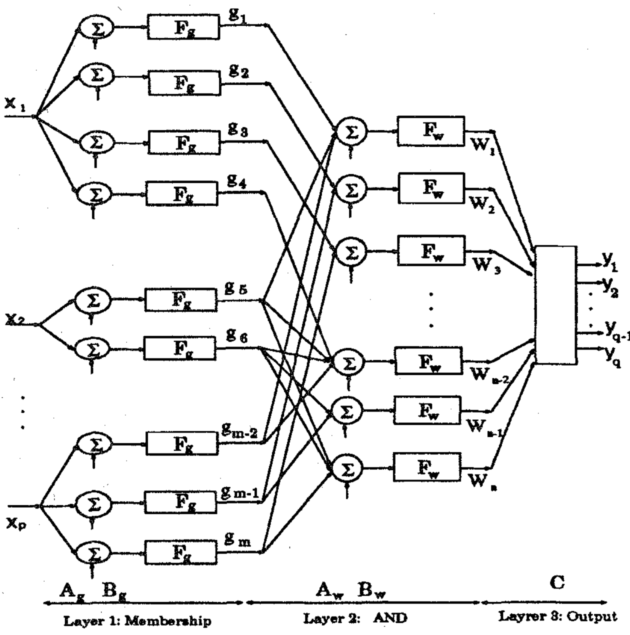


Fig. 2. Proposed neural network model for fuzzy rule-based systems.

TABLE I
SOME COMMONLY USED MEMBERSHIP FUNCTIONS (MFs)

MFs	Formula	Linear	Nonlinear
Linear	$1- \mid \frac{x-\mu}{\sigma} \mid$	$f = \frac{1}{\sigma}x - \frac{\mu}{\sigma}$	$1- \mid f \mid$
Gaussian	$exp\left(-\left(\frac{x-\mu}{\sigma}\right)^2\right)$	$f = \frac{1}{\sigma}x - \frac{\mu}{\sigma}$	e^{-f^2}
Cauchy	$\frac{1}{1+\left(\frac{x-\mu}{\sigma}\right)^2}$	$f = \frac{1}{\sigma}x - \frac{\mu}{\sigma}$	$\frac{1}{1+f^2}$
Sigmoid	$\frac{1}{1+exp\left(-\frac{x-\mu}{\sigma}\right)}$	$f = \frac{1}{\sigma}x - \frac{\mu}{\sigma}$	$\frac{1}{1+e^{-f}}$

IV. CONSTRUCTION OF PROPOSED NEURAL NETWORK MODEL

Fig. 2 shows the proposed neural fuzzy rule-based system. The system has three individual layers.

1) The first layer represents the parameters of membership functions in the antecedent part of the rule-base.
2) The second layer is a *soft-and* which is used as an approximation of a multiplication (or minimum) operation.
3) The third layer, i.e., output layer, contains the parameters of the consequent part of rule-base and defuzzification algorithm.

The functionality of each layer will be described in the following subsections.

A. Layer 1: Membership Functions

There are different choices for the shape of membership functions. Some membership functions commonly used in control applications are given in Table I.

We will decompose the membership function into two parts, a linear part and a nonlinear part. The linear part must be as a function of input(s).

$$h = au + b \tag{10}$$

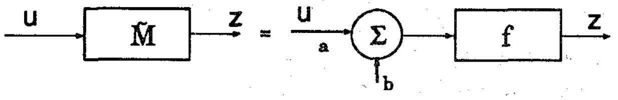


Fig. 3. An element of the first layer of the proposed neural network model.

and the nonlinear part yields the output of this subsystem,

$$z = f(h) \tag{11}$$

a and b are scalar constants, z and u are, respectively, the crisp output and the crisp input. $f(.)$ is the nonlinearity in the membership function. Fig. 3 illustrates the neural network model of a membership function.

When there are m individual membership functions, where, $m = \sum_{j=1}^{p} \hat{p}_j$, with p inputs, the formulation expressed in (11) can be converted to matrix formulation as follows:

$$g = F_g[\text{diag}\,(A_g)Hx + B_g] \tag{12}$$

where F_g is an m vector of nonlinear functions. Some commonly used nonlinearities are given in Table I. A_g and B_g are vectors of constant values with the general elements as indicated in (13). These elements are governed by the structure of the rules used in the FIS. H is a *connection matrix* with binary elements. This choice of binary values represents the structure of membership functions in the first layer. The vector g represents the output of all membership functions in the antecedent with the input vector x.

$$A_g = \begin{pmatrix} a_1^1 \\ \cdots \\ a_{\hat{p}_1}^1 \\ a_1^2 \\ \cdots \\ a_{\hat{p}_2}^2 \\ \cdots \\ a_1^p \\ \cdots \\ a_{\hat{p}_p}^p \end{pmatrix} H = \begin{pmatrix} e_{\hat{p}_1} \cdots 0 \\ \cdots\cdots\cdots \\ 0\ e_{\hat{p}_j}\ 0 \\ \cdots\cdots\cdots \\ 0 \cdots e_{\hat{p}_p} \end{pmatrix} B_g = \begin{pmatrix} b_1^1 \\ \cdots \\ b_{\hat{p}_1}^1 \\ b_1^2 \\ \cdots \\ b_{\hat{p}_2}^2 \\ \cdots \\ b_1^p \\ \cdots \\ b_{\hat{p}_p}^p \end{pmatrix} \tag{13}$$

where $e_{\hat{p}_j}$ is the vector of size, $\hat{p}_j$ $(j = 1 \cdots p)$ whose elements are all equal to one.

For specific examples illustrating how to convert the antecedent into a matrix form, please see Example 1 in Section VII.

B. Layer 2: Connective Operator AND

The connective operator "*and*" in most applications of fuzzy controller/decision is implemented by multiplication or by evaluating the minimum of the arguments. As indicated previously, this is a major stumbling block in the matrix formulation of a FIS. In order to obtain a matrix formulation of a fuzzy rule-based system, it is necessary to approximate the *and* operator. It is for this reason that we introduce a new operator, the *soft-and*, which utilizes a sigmoidal function.

$$\omega_i = \frac{1}{1 + \exp\left[-\left(\xi_0 + \sum_{k=1}^{\lambda} \xi_k g_k\right)\right]}. \tag{14}$$

This "soft and" is a single layer feedforward neural network (see Fig. 4) with λ inputs $(g_1, g_2, \cdots, g_\lambda)$ and one output w and fixed synaptic weights ξ_k: $k = 0 \cdots \lambda$. To determine the parameters ξ_k: $k = 0 \cdots \lambda$, which allow the neuron to best approximate multiplication, the following cost function must be minimized. Note

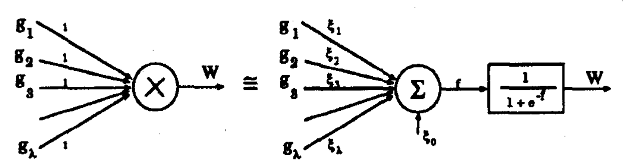


Fig. 4. An element of the second layer of the proposed neural network model.

TABLE II
SYNAPTIC WEIGHTS AND THRESHOLD VALUES FOR MULTIPLICATION NEURAL NETWORK MODEL

	ξ	ξ_0
Two inputs ($\lambda = 2$)	3.5856	-5.1741
Three inputs ($\lambda = 3$)	3.0223	-4.0948

that in this derivation, we impose the following condition: $0 \leq g_k \leq 1$ for $k = 1, \cdots, \lambda$.

$$e_w = \int_0^1 \cdots \int_0^1 \cdot \left\{ \prod_{k=1}^{\lambda} g_k - \frac{1}{1 + \exp\left[-\left(\xi_0 + \sum_{k=1}^{\lambda} \xi_k g_k \right) \right]} \right\}^2 \cdot dg_1 \cdots dg_\lambda. \tag{15}$$

Due to network symmetry, the synaptic weights $\xi_k = \xi$ for $k = 1 \cdots \lambda$. Therefore, only two parameters ξ and ξ_0 need to be determined. It should be noted that this approximation does not introduce a new set of unknown parameters to the system, as ξ and ξ_0 can be calculated prior to the construction of the model. The synaptic parameters for two ($\lambda = 2$) and three ($\lambda = 3$) inputs are shown in Table II.

When there are m inputs to the "soft and" operation with n outputs, the neural network model is the same as a SLP, which, in this case, is not fully connected. As we explained in Section III, the matrix formulation of the "soft and" operation can be represented as below:

$$w = F_w(A_w g + B_w) \tag{16}$$

where F_w is a n vector of sigmoid functions. A_w and B_w are constant matrices of dimensions $n \times m$ and $n \times 1$, respectively, representing the synaptic weights and thresholds as follows:

$$A_w = \xi A_I \tag{17}$$

$$B_w = \xi_0 e_n \tag{18}$$

where e_n is the vector of n in which all elements are equal to one and A_I is a matrix $n \times m$ whose elements are binary. An element one represents the existence of a connection between the jth input and the ith output. $j = 1, 2, \cdots, m$ and $i = 1, 2, \cdots, n$, and likewise a zero represents the lack of a connection.

C. Layer 3: Output Layer

The functionality of this layer is dependent upon the fuzzy if-then rules and the inference mechanism employed. However, in all the cases studied in this paper, this layer is a linear combination of the consequent parameters and the rule firing strength w. We will not give a special neural network model for this layer. It should be noted that it is possible to give a neural network model similar to the ones used for the first and the second layers, but since the purpose of this paper is to present a matrix formulation, it is not necessary to employ such a neural network structure.

V. MATRIX FORMULATION OF FUZZY INFERENCE SYSTEMS

A FIS can be viewed as a mapping from crisp input variables to crisp outputs governed by three sets of nonlinear equations as follows:

$$g(t) = f_1[g(t), x(t), t] \tag{19}$$

$$w(t) = f_2[w(t), g(t), t] \tag{20}$$

$$y(t) = f_3[y(t), w(t), t] \tag{21}$$

where $x(t) \in R^p$, $y(t) \in R^q$, are the inputs and outputs of the FIS. $g(t) \in R^m$ and $w(t) \in R^n$ are the firing weights and outputs of all individual membership functions in the antecedent part. There is no time dependency involved in the formulation of FIS, but in order to show explicit dependencies of the variables, we use the notation t. It may be thought of as the tth sample.

We wish to design a neural network model governed by the (19)–(21) such that the following error function is minimized.

$$J = \varepsilon^T \varepsilon \tag{22}$$

where $\varepsilon = y - y^d$, is a q dimensional vector, y^d and y are q dimensional vectors of the target function and the actual output, respectively.

We can discuss the set of (19)–(21) more specifically, if we know about the structure of the fuzzy if-then rules, the fuzzy reasoning, and the defuzzification methods. In the next three subsections, we will explain this matrix formulation for three different models discussed previously in Section II.

A. Matrix Formulation of a Semi-Fuzzy Model

Consider the following FIS matrix formulation which is exactly the same as the element by element formulation presented in Section II-Type I for the semi-fuzzy model.

$$g(t) = F_g[\mathrm{diag}\,(A_g)Hx(t) + B_g] \tag{23}$$

$$w(t) = F_w[A_w g(t) + B_w] \tag{24}$$

$$y(t) = [e_n^T w(t)]^{-1} C^T w(t) \tag{25}$$

$y(t) = [N_1', N_2', \cdots, N_\ell', \cdots, N_q']^T$ is a $q \times 1$ crisp output vector, $w(t)$ is a $n \times 1$ vector of firing rule strengths, $g(t)$ is a $m \times 1$ vector holding the outputs of the individual membership functions, and $x(t)$ is a $p \times 1$ input vector. A_g, H, and A_w are, respectively, matrices of dimensions $m \times 1$, $m \times p$, and $n \times m$ introduced previously in Section IV. B_g and B_w are, respectively, vectors of dimensions $m \times 1$ and $n \times 1$. F_g is a m dimensional vector of nonlinear functions. In the FIS situation, this is given by $F_g^T = [f_g(.) \quad f_g(.) \quad \cdots \quad f_g(.)]$ and $f_g(.)$ is one of the functions given in Table I. Similarly F_w is an n dimensional vector of sigmoidal functions. The consequent, C, is an $n \times q$ matrix whose elements (N_ℓ^i were introduced previously as consequent constant parameters) are given as follows:

$$C = \begin{pmatrix} N_1^1 & N_2^1 & \cdots & N_\ell^1 & \cdots & N_q^1 \\ N_1^2 & N_2^2 & \cdots & N_\ell^2 & \cdots & N_q^2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ N_1^i & N_2^i & \cdots & N_\ell^i & \cdots & N_q^i \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ N_1^n & N_2^n & \cdots & N_\ell^n & \cdots & N_q^n \end{pmatrix}. \tag{26}$$

The unknown parameters A_g, B_g, and C can be obtained by use of expert knowledge or by minimizing the error function J (22) using a gradient descent algorithm. The gradient of the error function with respect to the parameters A_g, B_g, and C are given by:

$$\frac{\partial J}{\partial C} = 2[e_n^T w(t)]^{-1} w(t)\varepsilon^T(t) \tag{27}$$

$$\frac{\partial J}{\partial B_g} = 2[e_n^T w(t)]^{-1} \Lambda_g \delta [C - e_n y^T(t)]\varepsilon(t) \tag{28}$$

$$\frac{\partial J}{\partial A_g} = 2[e_n^T w(t)]^{-1} \operatorname{diag}[Hx(t)] \cdot \Lambda_g \delta [C - e_n y^T(t)]\varepsilon(t) \tag{29}$$

$$\delta = A_w^T \Lambda_w \tag{30}$$

where Λ_w denotes an $n \times n$ diagonal matrix, whose diagonal elements are given by $f'_w(.)$, denoting the first derivative of the nonlinear function $f_w(.) \in F_w$. Similarly, Λ_g is an $m \times m$ diagonal matrix whose diagonal elements are given by $f'_g(.)$, the first derivative of the nonlinear function $f_g(.) \in F_g$.

The application of (27)–(30) for a gradient descent minimization algorithm will be given in Section VI.

B. Matrix Formulation of Takagi and Sugeno's Model

In the semi-fuzzy model explained in the previous subsection, the consequent part of rules are constants. Takagi and Sugeno introduced their model whereby the consequent part is a linear combination of the crisp inputs. The matrix formulation for this specific form of fuzzy if-then rules is as follows:

$$g(t) = F_g[\operatorname{diag}(A_g)Hx(t) + B_g] \tag{31}$$

$$w(t) = F_w[A_w g(t) + B_w] \tag{32}$$

$$y(t) = [e_n^T w(t)]^{-1}\{C + D[I_q \otimes x(t)]\}^T w(t). \tag{33}$$

Matrix D is defined as a column matrix whose elements are parameters in the consequent part which are expressed as a linear combination of inputs. I_q is a unit matrix of order $q \times q$ and $\otimes$ denotes the Kronecker product [6].

$$D = [D_1 \mid D_2 \mid \cdots \mid D_\ell \mid \cdots D_q]. \tag{34}$$

The matrix D_j is defined as follows:

$$D_\ell = \begin{pmatrix} (\hat{N}_1^1)_\ell & (\hat{N}_2^1)_\ell & \cdots & (\hat{N}_j^1)_\ell & \cdots & (\hat{N}_p^1)_\ell \\ (\hat{N}_1^2)_\ell & (\hat{N}_2^2)_\ell & \cdots & (\hat{N}_j^2)_\ell & \cdots & (\hat{N}_p^2)_\ell \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ (\hat{N}_1^i)_\ell & (\hat{N}_2^i)_\ell & \cdots & (\hat{N}_j^i)_\ell & \cdots & (\hat{N}_p^i)_\ell \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ (\hat{N}_1^n)_\ell & (\hat{N}_2^n)_\ell & \cdots & (\hat{N}_j^n)_\ell & \cdots & (\hat{N}_p^n)_\ell \end{pmatrix}. \tag{35}$$

The corresponding gradients for minimizing the error function (22) are given in the Appendix.

C. Matrix Formulation of General Fuzzy Model

When both the antecedent and the consequent of rules are expressed in fuzzy value form, we have a set of general fuzzy if-then rules. With the help of the defuzzification method explained in Section II, it is possible to construct our proposed matrix formulation. Due to the complexity of the matrix formulation in this case, we will give the formulation only for systems with one output i.e., $q = 1$. The case for $q \neq 1$ can be obtained in a similar fashion. The proposed formulation is given below:

$$g(t) = F_g[\operatorname{diag}(A_g)Hx(t) + B_g] \tag{36}$$

$$\Omega(t) = QF_w[A_w g(t) + B_w] \tag{37}$$

$$y(t) = [D^T\Omega(t)]^{-1} C^T \operatorname{diag}(D)\,\Omega(t). \tag{38}$$

Equations (36) and (37) are explained previously. In (38) the firing weight, $w(t)$, is substituted with $\Omega(t)$, which is related to $w(t)$ by the operator matrix Q as follows:

$$\Omega(t) = Qw(t). \tag{39}$$

The operator Q is a $\hat{q}_\ell \times n$ matrix with binary elements that define the connection of different firing weights w_i to different individual consequent fuzzy values. If an element is one in particular row of the operator matrix Q, it means that the consequent membership functions for those rules must be equal. Vectors C and D are given, respectively, as follows:

$$C = \begin{pmatrix} N_\ell^1 \\ N_\ell^2 \\ \cdots \\ N_\ell^{\hat{q}_\ell} \end{pmatrix} \quad D = \begin{pmatrix} S_\ell^1 \\ S_\ell^2 \\ \cdots \\ S_\ell^{\hat{q}_\ell} \end{pmatrix}. \tag{40}$$

The corresponding gradients for minimizing the error function are given in the Appendix.

It should be noted that the gradients for the consequent membership functions are expressed in terms of the parameters describing the center of the membership functions (vector C) and the area of individual membership functions in the consequent part (vector D).

VI. The Gradient Descent Training Algorithm

The vector of membership function parameters for the antecedent A_g, B_g, and the vector of constant variables for the consequent C and D can be adjusted to minimize an objective function J_{tot} defined as follows:

$$J_{tot} = \sum_{p=1}^{P} J_p = \sum_{p=1}^{P} \varepsilon_p^T \varepsilon_p \tag{41}$$

where J_p is the error function for the pth training data [defined earlier in (22)]. The number of exemplars in the training data set is P.

To update the unknown parameters in matrices A_g, B_g, C, and D, we introduce a steepest descent method [4], [5] to minimize the total error function J_{tot}.

$$A_g(B_g, C, D)|_{new} = -\eta \frac{\partial J_{tot}}{\partial A_g(B_g, C, D)} + \alpha A_g(B_g, C, D)|_{old} \tag{42}$$

where η is learning rate which can be expressed as follows:

$$\eta = \frac{\kappa}{\sqrt{\left(\frac{\partial J_{tot}}{\partial D}\right)^2 + \left(\frac{\partial J_{tot}}{\partial C}\right)^2 + \left(\frac{\partial J_{tot}}{\partial B_g}\right)^2 + \left(\frac{\partial J_{tot}}{\partial A_g}\right)^2}}. \tag{43}$$

Note that in this case, the learning is dependent on the "strength" of the gradients. If the gradients are large, then the corresponding learning rate is small. On the other hand, if the gradients are small, learning rate can be increased. The constant parameter α is the *momentum* term of the gradient descent method. The constant κ is the corresponding step size. The gradient vectors $(\partial J_{tot}/\partial D)$, $(\partial J_{tot}/\partial C)$, $(\partial J_{tot}/\partial B_g)$, and $(\partial J_{tot}/\partial A_g)$ are defined as follows:

$$\frac{\partial J_{tot}}{\partial A_g(B_g, C, D)} = \sum_{p=1}^{P} \frac{\partial J_p}{\partial A_g(B_g, C, D)}. \tag{44}$$

Due to the nonlinear interaction in the first layer between A_g and B_g, the training algorithm does not perform very well. If the step size κ is chosen to be very small, it performs relatively well but then the speed is very slow. To utilize the nonlinear interaction in the first layer among the parameters, we introduce two vectors $\overline{A}_g$ and $\overline{B}_g$ which can be obtained by a simple operation on the vectors of A_g and B_g, respectively.

$$\begin{aligned}\overline{A}_g &= \mathrm{diag}\,[\mathrm{diag}\,(A_g)^{-1}] \leftrightarrow A_g \\ &= \mathrm{diag}[\mathrm{diag}(\overline{A}_g)^{-1}] \end{aligned} \tag{45}$$

$$\begin{aligned}\overline{B}_g &= -\mathrm{diag}\,(A_g)^{-1} B_g \leftrightarrow B_g \\ &= -\mathrm{diag}\,(\overline{A}_g)^{-1}\overline{B}_g.\end{aligned} \tag{46}$$

Therefore, the gradient vectors for the new parameter vectors can be achieved by using the gradients of A_g and B_g as follows:

$$\frac{\partial J_p}{\partial \overline{B}_g} = -\mathrm{diag}\,(A_g)\,\frac{\partial J_p}{\partial B_g} \tag{47}$$

$$\frac{\partial J_p}{\partial \overline{A}_g} = -\mathrm{diag}\,(A_g) \cdot \left[\mathrm{diag}\,(A_g)\,\frac{\partial J_p}{\partial A_g} + \mathrm{diag}\,(B_g)\,\frac{\partial J_p}{\partial B_g}\right]. \tag{48}$$

Based on the new gradients and the new parameter vectors, we can expressed the training algorithm for vectors $\overline{A}_g$ and $\overline{B}_g$ as follows:

$$\overline{A}_g|_{new} = -\eta \frac{\partial J_{tot}}{\partial \overline{A}_g} + \alpha \overline{A}_g|_{old} \tag{49}$$

$$\overline{B}_g|_{new} = -\eta \frac{\partial J_{tot}}{\partial \overline{B}_g} + \alpha \overline{B}_g|_{old}. \tag{50}$$

Similarly, the learning rate η can be calculated as follows;

$$\eta = \frac{\kappa}{\sqrt{\left(\frac{\partial J_{tot}}{\partial D}\right)^2 + \left(\frac{\partial J_{tot}}{\partial C}\right)^2 + \left(\frac{\partial J_{tot}}{\partial \overline{B}_g}\right)^2 + \left(\frac{\partial J_{tot}}{\partial \overline{A}_g}\right)^2}}. \tag{51}$$

VII. Illustrative Example

To illustrate the matrix formulation expressed in previous sections, we consider a fuzzy rule-based system with 2 inputs ($p = 2$), one output ($q = 1$), six rules ($n = 6$), three individual membership functions for the first input ($\hat{p}_1 = 3$), and two individual membership functions for the second input ($\hat{p}_2 = 2$). The membership functions in the first layer are chosen to be of a Gaussian shape. The neural network model for this system is shown in Fig. 5.

A computer simulation program is written to train the fuzzy system with matrix formulations. The training data y^d is obtained from a nonlinear surface given by (52). The universes of inputs U_1 and U_2 for both x_1 and x_2 are $[-5 \quad 5]$.

$$y^d = 10e^{-(x_1/4)^2} e^{-(x_2-4/4)^2}. \tag{52}$$

Based on some intuitive understanding from the desired surface given by (52), we assign initial values for the vectors A_g, B_g, C, and D. With an appropriate combination of step size κ and the momentum α, the algorithm converges.

In order to evaluate the performance of different adaptive networks, we define an *average percentage error* (APE) as follows [9]:

$$APE = \frac{\sum_{p=1}^{P} J_p}{\sum_{p=1}^{P} y_p^d} \times 100\% \tag{53}$$

where in our simulation we use $P = 441$ (21×21) sampling points. The various matrix formulations expressed in Section V will be illustrated with the given data.

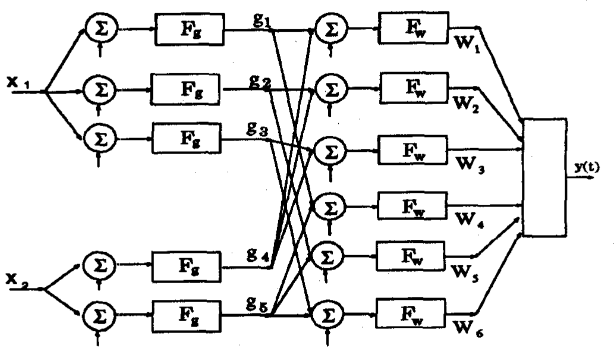


Fig. 5. Neural network model for a fuzzy rule-based system with two inputs, one output and six rules.

Example 1: At first we consider a semi-fuzzy model for the rule-based system. Based on this model, the rules can be expressed as follows:

Rule 1: If x_1 is $\tilde{M}_1^1$ and x_2 is $\tilde{M}_2^1$ then y is N_1^1, else
Rule 2: If x_1 is $\tilde{M}_1^2$ and x_2 is $\tilde{M}_2^1$ then y is N_1^2, else
Rule 3: If x_1 is $\tilde{M}_1^3$ and x_2 is $\tilde{M}_2^1$ then y is N_1^3, else
Rule 4: If x_1 is $\tilde{M}_1^1$ and x_2 is $\tilde{M}_2^2$ then y is N_1^4, else
Rule 5: If x_1 is $\tilde{M}_1^2$ and x_2 is $\tilde{M}_2^2$ then y is N_1^5, else
Rule 6: If x_1 is $\tilde{M}_1^3$ and x_2 is $\tilde{M}_2^2$ then y is N_1^6

The initial values for the unknown matrices A_g, B_g, C, H, A_w, and B_w are given as follows:

$$A_g = \begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_1^2 \\ a_2^2 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} B_g = \begin{pmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \\ b_1^2 \\ b_2^2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 1 \\ -1 \end{pmatrix}$$

$$C = \begin{pmatrix} N_1^1 \\ N_1^2 \\ N_1^3 \\ N_1^4 \\ N_1^5 \\ N_1^6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.5 \\ 0 \\ 1 \\ 10 \\ 1 \end{pmatrix} H = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$A_w = 3.5856 \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} B_w = -5.1741 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{54}$$

The average percentage errors for the first 500 epochs of training with a step size $\kappa = 0.01$, and a momentum $\alpha = 0.95$ are plotted in Fig. 6(a). The vector of unknown parameters after training has stopped are given as follows:

$$A_g = \begin{pmatrix} 0.2414 \\ 0.0001 \\ 0.2414 \\ 0.2365 \\ 0.2120 \end{pmatrix} B_g = \begin{pmatrix} 1.2301 \\ 0.0000 \\ -1.2301 \\ 1.0447 \\ -1.6591 \end{pmatrix} C = \begin{pmatrix} 0.0667 \\ 1.1354 \\ 0.0667 \\ 0.8076 \\ 10.2257 \\ 0.8076 \end{pmatrix}. \tag{55}$$

This simulation was repeated for training using the modified training algorithms i.e., using vectors of $\overline{A}_g$ and $\overline{B}_g$. The APE is shown in Fig. 6(b) where the step size is $\kappa = 0.1$ and the momentum $\alpha = 0$.

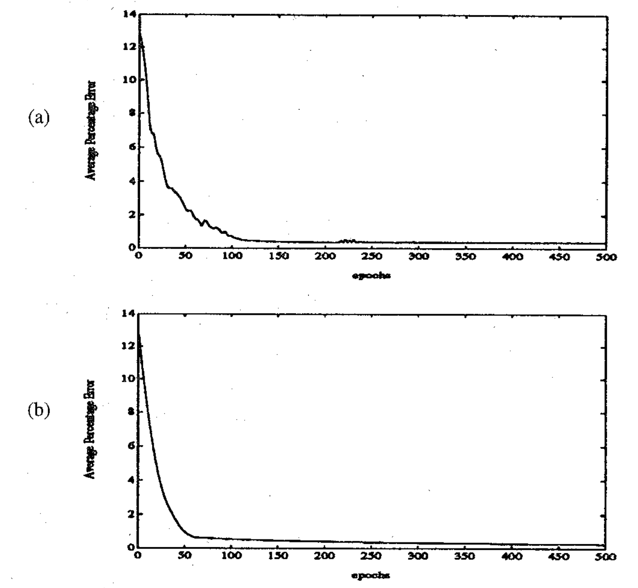


Fig. 6. Average percentage error for semi-fuzzy model. (a) Training algorithm with step size $\kappa = 0.01$ and momentum $\alpha = 0.95$. (b) Modified training algorithm with step size $\kappa = 0.1$ and momentum $\alpha = 0$.

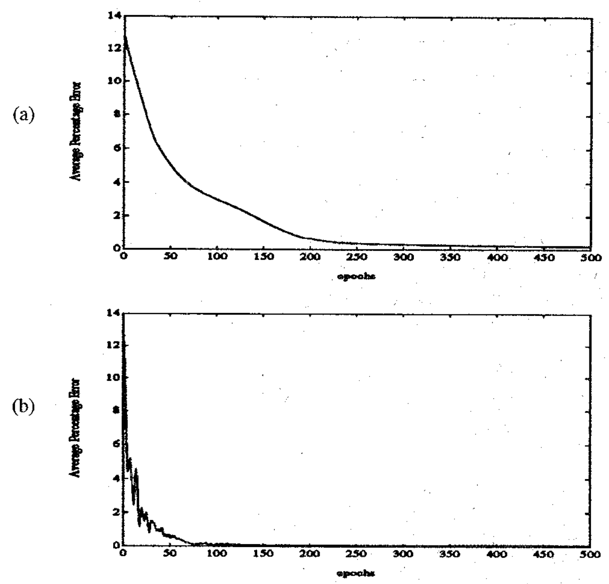


Fig. 7. Average percentage error for Sugeno fuzzy if-then rule. (a) Training algorithm with step size $\kappa = 0.01$ and momentum $\alpha = 0$. (b) Modified training algorithm with step size $\kappa = 0.1$ and momentum $\alpha = 0.95$.

Example 2: The matrix formulation for the same inputs and output as indicated in Example 1 can be considered for a rule-based system with Sugenos's fuzzy if-then rules.

Rule 1: If x_1 is $\tilde{M}_1^1$ and x_2 is $\tilde{M}_2^1$ then
$y = N_1^1 + (\hat{N}_1^1)_1 x_1 + (\hat{N}_2^1)_1 x_2$, else

Rule 2: If x_1 is $\tilde{M}_1^2$ and x_2 is $\tilde{M}_2^1$ then
$y = N_1^2 + (\hat{N}_1^2)_1 x_1 + (\hat{N}_2^2)_1 x_2$, else

Rule 3: If x_1 is $\tilde{M}_1^3$ and x_2 is $\tilde{M}_2^1$ then
$y = N_1^3 + (\hat{N}_1^3)_1 x_1 + (\hat{N}_2^3)_1 x_2$, else

Rule 4: If x_1 is $\tilde{M}_1^1$ and x_2 is $\tilde{M}_2^2$ then
$y = N_1^4 + (\hat{N}_1^4)_1 x_1 + (\hat{N}_2^4)_1 x_2$, else

Rule 5: If x_1 is $\tilde{M}_1^2$ and x_2 is $\tilde{M}_2^2$ then
$y = N_1^5 + (\hat{N}_1^5)_1 x_1 + (\hat{N}_2^5)_1 x_2$, else

Rule 6: If x_1 is $\tilde{M}_1^3$ and x_2 is $\tilde{M}_2^2$ then
$y = N_1^6 + (\hat{N}_1^6)_1 x_1 + (\hat{N}_2^6)_1 x_2$

The matrices A_g, B_g, and C, A_w, and B_w are as defined as in Example 1. The initial values for the unknown matrix D is defined as follows:

$$D = \begin{pmatrix} (\hat{N}_1^1)_1 & (\hat{N}_2^1)_1 \\ (\hat{N}_1^2)_1 & (\hat{N}_2^2)_1 \\ (\hat{N}_1^3)_1 & (\hat{N}_2^3)_1 \\ (\hat{N}_1^4)_1 & (\hat{N}_2^4)_1 \\ (\hat{N}_1^5)_1 & (\hat{N}_2^5)_1 \\ (\hat{N}_1^6)_1 & (\hat{N}_2^6)_1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}. \tag{56}$$

The average percentage errors for the first 500 epochs of training are plotted in Fig. 7(a) where the step size $\kappa = 0.01$ and momentum $\alpha = 0$. The vector of parameters after training has stopped is given in (57). In Fig. 7(b), the APE is plotted for the modified training algorithm where the step size $\kappa = 0.1$ and the momentum $\alpha = 0.95$.

$$A_g = \begin{pmatrix} 0.3131 \\ 0.2502 \\ 0.3131 \\ 0.3479 \\ 0.1783 \end{pmatrix} \quad B_g = \begin{pmatrix} 1.4631 \\ 0.0000 \\ -1.4631 \\ 0.2932 \\ -0.6537 \end{pmatrix}$$

$$C = \begin{pmatrix} 0.2133 \\ 0.7546 \\ 0.2133 \\ 0.8687 \\ 10.0288 \\ 0.8687 \end{pmatrix} \quad D = \begin{pmatrix} -0.2357 & 0.0666 \\ 0.0000 & 0.8261 \\ 0.2357 & 0.0666 \\ 0.5012 & -0.1101 \\ 0.0000 & 1.2781 \\ -0.5012 & -0.1101 \end{pmatrix}. \tag{57}$$

Example 3: Consider the rule base system given in Example 1, but with the fuzzy values in the consequent part using instead, 4 individual membership functions for the output variable.

Rule 1: If x_1 is $\tilde{M}_1^1$ and x_2 is $\tilde{M}_2^1$ then y is $\tilde{N}_1^1$, else
Rule 2: If x_1 is $\tilde{M}_1^2$ and x_2 is $\tilde{M}_2^1$ then y is $\tilde{N}_1^2$, else
Rule 3: If x_1 is $\tilde{M}_1^3$ and x_2 is $\tilde{M}_2^1$ then y is $\tilde{N}_1^1$, else
Rule 4: If x_1 is $\tilde{M}_1^1$ and x_2 is $\tilde{M}_2^1$ then y is $\tilde{N}_1^4$, else
Rule 5: If x_1 is $\tilde{M}_1^2$ and x_2 is $\tilde{M}_2^1$ then y is $\tilde{N}_1^3$, else
Rule 6: If x_1 is $\tilde{M}_1^3$ and x_2 is $\tilde{M}_2^1$ then y is $\tilde{N}_1^4$

The constant matrix Q and initial values for unknown vectors C, and D are given as follows:

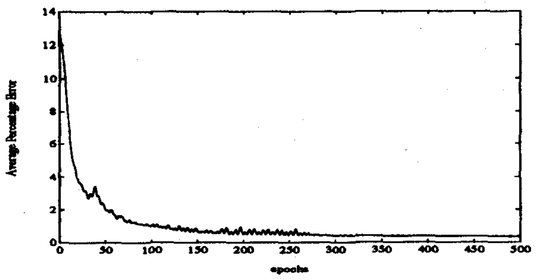
$$Q = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} N_1^1 \\ N_1^2 \\ N_1^3 \\ N_1^4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0.5 \\ 10 \\ 1 \end{pmatrix} D = \begin{pmatrix} S_1^1 \\ S_1^2 \\ S_1^3 \\ S_1^4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{58}$$

The average percentage errors for the first 500 epochs of training are shown in Fig. 8(a) where the step size $\kappa = 0.01$ and the momentum $\alpha = 0.95$. The vector of parameters after training is given in (59). In Fig. 8(b), the APE is shown for the modified training algorithm where the step size $\kappa = 0.01$ and the momentum $\alpha = 0.95$.

$$A_g = \begin{pmatrix} 0.2212 \\ 0.1792 \\ 0.2212 \\ 0.2070 \\ 0.2211 \end{pmatrix} \quad B_g = \begin{pmatrix} 1.3526 \\ 0.0000 \\ -1.3526 \\ 1.1388 \\ -1.8696 \end{pmatrix}$$

(a)



(b)

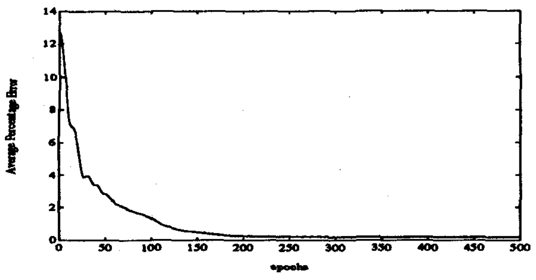


Fig. 8. Average percentage error for general fuzzy model. (a) Training algorithm with step size $\kappa = 0.01$ and momentum $\alpha = 0.95$ (b) Modified training algorithm with step size $\kappa = 0.01$ and momentum $\alpha = 0.95$.

$$C = \begin{pmatrix} 0.3832 \\ -2.9100 \\ 17.1048 \\ 1.2079 \end{pmatrix} D = \begin{pmatrix} 1.8194 \\ 0.8991 \\ 1.0216 \\ 2.1085 \end{pmatrix}. \tag{59}$$

VIII. Conclusions

In this paper we have introduced a matrix formulation for fuzzy rule-based systems in a very general and commonly used structure with p inputs, q outputs, and n rules. It has been shown that a fuzzy rule-based system with some commonly used mechanism of reasoning can be expressed by a set of nonlinear and linear matrix equations. Subsequently the training algorithm for adaptive membership functions are also provided in matrix formulations. Illustrative examples are given to clarify the notation used and the applicability of the proposed method.

Appendix

Gradients for minimizing the error function for Takagi and Sugeno's model can be expressed in matrix formulation as follows:

$$\begin{aligned}
\frac{\partial J}{\partial C} &= 2[e_n^T w(t)]^{-1} w(t)\varepsilon^T(t) \\
\frac{\partial J}{\partial D} &= 2[e_n^T w(t)]^{-1} w(t)\varepsilon^T(t)[I_q \otimes x^T(t)] \\
\frac{\partial J}{\partial B_g} &= 2[e_n^T w(t)]^{-1} \\
&\quad \cdot \Lambda_g \delta \{C + D[I_q \otimes x(t)] - e_n y^T(t)\}\varepsilon(t) \\
\frac{\partial J}{\partial A_g} &= 2[e_n^T w(t)]^{-1} \operatorname{diag}[Hx(t)] \\
&\quad \cdot \Lambda_g \delta \{C + D[I_q \otimes x(t)] - e_n y^T(t)\}\varepsilon(t) \\
\delta &= A_w^T \Lambda_w.
\end{aligned}$$

The gradients for General fuzzy model are:

$$\begin{aligned}
\frac{\partial J}{\partial C} &= 2[D^T \Omega(t)]^{-1} \operatorname{diag}(D)\Omega(t)\varepsilon(t) \\
\frac{\partial J}{\partial D} &= 2[D^T \Omega(t)]^{-1} \operatorname{diag}[\Omega(t)][C - e_{\hat{q}\ell} y(t)]\varepsilon(t) \\
\frac{\partial J}{\partial B_g} &= 2[D^T \Omega(t)]^{-1} \\
&\quad \cdot \Lambda_g \delta \operatorname{diag}(D)[C - e_{\hat{q}\ell} y(t)]\varepsilon(t) \\
\frac{\partial J}{\partial A_g} &= 2[D^T \Omega(t)]^{-1} \operatorname{diag}[Hx(t)] \\
&\quad \cdot \Lambda_g \delta \operatorname{diag}(D)[C - e_{\hat{q}\ell} y(t)]\varepsilon(t) \\
\delta &= A_w^T \Lambda_w Q^T.
\end{aligned}$$

References

[1] A. Athalye, D. Edwards, and V. S. Manoranjan, "On design a fuzzy control system using an optimization algorithm," *Fuzzy Sets Syst.*, vol. 56, pp. 281–290, 1993.

[2] J. Bruske, E. V. Puttkamer, and U. R. Zimmer, "Spin-nfds learning and preset knowledge for surface fusion—A neural fuzzy decision system," in *Proc. Australia and New Zealand Conf. Intelligent Information Systems (ANZIIS-93)*, Perth, Australia, Dec. 1993, pp. 396–401.

[3] C. Darken and J. Moody, "Connectionist models summer school," in *Proc. 1988 Connectionist Models Summer School*, D. S. Touretzky, T. Sejnowski, and G. E. Hinton, Eds.. San Francisco: Morgan Kaufmann, 1989.

[4] R. Fletcher, *Practical Methods of Optimization*, 2nd ed. New York: Wiley, 1987.

[5] P. E. Gill, *Practical Optimization.* London: Academic, 1981.

[6] A. Graham, *Kronecker Products and Matrix Calculus With Applications.* New York: Halsted, 1981.

[7] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation.* Reading, MA: Addison Wesley, 1990.

[8] S. Horikawa, T. Furuhashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 801–806, Sept. 1992.

[9] J. S. R. Jang, "Anfis: Adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 3, pp. 665–685, May/June 1993.

[10] ——, "Self-learning fuzzy controllers based on temporal back propagation," *IEEE Trans. Syst., Man, Cybern.*, vol. 3, no. 5, pp. 714–723, Sept. 1992.

[11] T. Kailath, *Linear Systems.* Englewood Cliffs, NJ: Prentice Hall, 1980.

[12] C. T. Lin and C. S. George Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Comput.*, vol. 40, no. 12, pp. 1320–1336, Dec. 1991.

[13] S. Liu and S. Hu, "A method of generating control rule model and its application," *Fuzzy Sets Syst.*, vol. 52, pp. 33–37, 1992.

[14] A. Lotfi and A. C. Tsoi, "Importance of membership functions: A comparative study on different learning methods for fuzzy inference systems," in *Proc. Third IEEE Int. Conf. Fuzzy Systems*, Orlando, FL, June 1994, pp. 1791–1796.

[15] ——, "Learning fuzzy inference systems using an adaptive membership function scheme," *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, no. 2, pp. 326-331, Apr. 1996.

[16] E. H. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *Int. J. Man Machine Studies*, vol. 7, no. 1, pp. 1–13, 1974.

[17] E. H. Mamdani, "Twenty years of fuzzy control: Experiences gained and lesson learnt," in *Proc. IEEE Conf. Fuzzy Systems*, San Francisco, CA, Apr. 1993, vol. 1, pp. 339–344.

[18] M. Mizumoto, "Fuzzy control under various reasoning method," *Inform. Sci.*, vol. 45, pp. 129–151, 1988.

[19] K. Nishimori, H. Tokutaka, and S. Hirakawa, "Comparison of several fuzzy reasoning methods on driving control of a model car," in *Proc. 2nd Int. Conf. Fuzzy Logic and Neural Networks*, Iizuka, Japan, July 1992, pp. 421–424.

[20] H. Nomura, I. Hayashi, and N. Wakami, "A learning method of fuzzy inference rules by descent method," in *Proc. IEEE Int. Conf. Fuzzy Systems*, Mar. 1992, pp. 203–210.

[21] P. J. Pacini and B. Kosko, "Adaptive fuzzy systems for target tracking," *Intell. Syst. Eng.*, pp. 3–21, July 1992.

[22] B. G. Song *et al.*, "Adaptive membership function fusion and annihilation in fuzzy if-then rules," in *Proc. IEEE Int. Conf. Fuzzy Systems*, Apr. 1993, pp. 961–967.

[23] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 1, pp. 116–132, Jan. 1985.

[24] A. C. Tsoi and A. Back, "A new dynamic neuron model, and its training algorithm," preprint, 1994.

[25] A. C. Tsoi, D. S. C. So, and A. Sergejew, "Classification of electroencephalogram using artificial neural networks," in *Neural Information Processing Systems,* J. Cowan, L. Giles, and G. Tesauro, Eds. San Francisco: Morgan Kaufmann, 1994, vol. 6.
[26] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Syst., Man, Cybern.,* vol. 22, no. 6, pp. 1414–1427, Dec. 1992.
[27] ——, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks,* vol. 3, no. 5, pp. 807–814, Sept. 1992.

Hierarchical Reduction and Partition of Hypergraph

Hyung Lee-Kwang and Choong Ho Cho

***Abstract*—In this paper, a hierarchical reduction method of hypergraphs is proposed. A macro-vertex in a reduced hypergraph corresponds to an edge of the original hypergraph, and thus a reduced hypergraph can provide a partition of a system. The reduction is realized by the iterations and the sequence of hierarchical reduction gives a sequence of hierarchical partitions. The proposed method allows to reduce and decompose the complexity of the system represented by hypergraphs.**

I. Introduction

The hypergraph was introduced by Berge [1] and has been considered as a useful tool to analyze the structure of a system and to model a partition, covering and clustering [9]. However, the complexity of the model is increased with the number of vertices and edges in the hypergraph. This complexity often gives rise to considerable errors on modeling and analyzing the system. The analysis of a large-scale system is faced with the complexity, and it is convenient to analyze with reduced models and submodels.

Reduction and decomposition of system dimension is a general approach to manage the complexity in the field of system engineering and large-scale systems [5], [7], [10]. Our fundamental principle in managing the complexity is to reduce the size of edges and vertices by hypergraph reduction [11], [12].

The hypergraph reduction is a procedure that homomorphically transforms hypergraphs to their reduced graphs. The procedure reduces the size of edges and vertices and thus the complexity of models. However, it has been pointed out that there is no approach to reduce the hypergraphs.

Therefore, in this paper, a reduction method of hypergraphs is proposed. In the reduction, an edge is reduced into a macro-vertex. The reduction is realized by iterations and the iterations provide a sequence of hierarchical reductions. A macro-vertex in a reduced hypergraph represents an edge (sub-hypergraph) and thus a reduced hypergraph can give a partition of a system. The sequence of reduction can also provide a sequence of partitions. The partitions are ordered by the inclusion relation. The sequence of partition gives hierarchical partitions of the system.

Manuscript received Aril 24, 1994; revised December 28, 1994.

H. Lee-Kwang is with the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Taejon 305-701, Seoul, Korea (e-mail: khlee@monami.kaist.ac.kr).
C. H. Cho is with the Department of Computer Science, Korea University, Chochiwon, Choongnam 393-800, Korea.

Publisher Item Identifier S 1083-4419(96)02310-2.

In Section II, a brief review on the hypergraph is given and Section III provides a hierarchical reduction method of hypergraphs. Section IV shows an example of a hierarchical reductions and hierarchical partitions of a hypergraph.

II. Hypergraph

The hypergraph $H = (V, \mathcal{E})$ was proposed by Berge [1] and is defined as follows:

$$\begin{aligned} &H = (V, \mathcal{E}) \text{ where} \\ &V = \{x_1, x_2, \cdots, x_n\} : \text{a finite set of vertices} \\ &\mathcal{E} = \{E_1, E_2, \cdots, E_m\} : \text{a family of subsets of } V \\ &\qquad E_j \neq \phi, \quad j = 1, \cdots, m \\ &\qquad \cup_j E_j = V. \end{aligned}$$

The set V is called the set of vertices and $\mathcal{E}$ is the set of edges (or hyperedges). In the diagram, the edge E_j is represented by a solid line surrounding its vertices if $|E_j| \geq 2$; if $|Ej| = 1$ by a cycle on the element. If $|E_j| = 2$ for all j, the hypergraph becomes an ordinary (undirected) graph. The hypergraph $(V, \mathcal{E})$ can be also represented by $(V; E_1, E_2, \cdots, E_m)$. The order of a hypergraph is the cardinality of set V, that is, $|V|$.

In a hypergraph, two vertices x and y are said to be adjacent if there exists an edge E_j which contains the two vertices ($x \in E_j$, $y \in E_j$). Two edges E_i and E_j are said to be adjacent if their intersection is not empty ($E_i \cap E_j \neq \phi$, $i \neq j$).

In a hypergraph $H = (V, \mathcal{E})$; $V = \{x_1, x_2, \cdots, x_n\}$ and $\mathcal{E} = \{E_1, E_2, \cdots, E_m\}$, its incidence matrix is a matrix $M_H = (a_{ij})_{n \times m}$ with m columns representing the edges and n rows representing the vertices, where the elements a_{ij} indicate the membership of vertex to hyperedge as follows:

$$\begin{aligned} a_{ij} &= 1 \quad \text{if} \quad x_i \in E_j \\ &= 0 \quad \text{if} \quad x_i \notin E_j. \end{aligned}$$

For example, consider a hypergraph $H = (V, \mathcal{E})$ such that

$$\begin{aligned} V &= \{x_1, x_2, x_3, x_4, x_5\} \\ \mathcal{E} &= \{E_1, E_2, E_3\} \\ E_1 &= \{x_1, x_2\}, \\ E_2 &= \{x_2, x_3, x_4\}, \\ E_3 &= \{x_4, x_5\}. \end{aligned}$$

The hypergraph can be shown as in Fig. 1 and its incidence matrix is as follows:

	E_1	E_2	E_3
x_1	1	0	0
x_2	1	1	0
x_3	0	1	0
x_4	0	1	1
x_5	0	0	1

In general, a hypergraph represents a covering of set V. In a hypergraph, if every vertex has its degree 1 (i.e., $E_i \cap E_j = \phi$, $i \neq j$), the hypergraph represents a partition of V.