Neural-Network-Based Fuzzy Logic Control and Decision System

Chin-Teng Lin, *Student Member, IEEE*, and C. S. George Lee, *Senior Member, IEEE*

Abstract—**A general neural-network (connectionist) model for fuzzy logic control and decision/diagnosis systems is proposed. The proposed connectionist model can be contrasted with the traditional fuzzy logic control and decision system in their network structure and learning ability. Such fuzzy control/decision networks can be constructed from training examples by machine learning techniques, and the connectionist structure can be trained to develop fuzzy logic rules and find optimal input/output membership functions. By combining both unsupervised (self-organized) and supervised learning schemes, the learning speed converges much faster than the original backpropagation learning algorithm. This connectionist model also provides human-understandable meaning to the normal feedforward multilayer neural network in which the internal units are always opaque to the users. The connectionist structure also avoids the rule-matching time of the inference engine in the traditional fuzzy logic system. Two examples are presented to illustrate the performance and applicability of the proposed connectionist model.**

Index Terms— **Connectionist, distributed representation, defuzzifier, fuzzifier, fuzzy logic rule, membership function, receptive field, supervised and unsupervised learning.**

I. Introduction

During the past decade, fuzzy logic [1] has found a variety of applications in various fields ranging from industrial process control [2], [3] to medical diagnosis [4] and securities trading [5]. Most notably, a fuzzy logic system has been applied to control nonlinear, time-varying, ill-defined systems [2], to control systems whose dynamics are exactly known, such as servomotor position control [6] and robot arm control [7], and to manage complex decision-making or diagnosis systems [4], [5].

Fuzzy logic systems base their decisions on inputs in the form of linguistic variables derived from membership functions which are formulas used to determine the fuzzy set to which a value belongs and the degree of membership in that set. The variables are then matched with the preconditions of linguistic IF–THEN rules (fuzzy logic rules), and the response of each rule is obtained through *fuzzy implication*. To perform compositional rule of inference, the response of each rule is weighted according to the confidence or degree of membership of its inputs, and the centroid of the responses is calculated to generate the appropriate output. At present, there is no systematic procedure for the design of fuzzy logic systems. The most straightforward approach is to define membership functions and rules subjectively by studying a human-operated system or an existing controller and then testing the design for the proper output. The membership functions and/or rules should be adjusted if the design fails the test. Recent direction of exploration is to design fuzzy logic systems that have the capability to learn from experience itself. Currently, only some published results have the ability to create fuzzy control rules and to modify them based on experience [7], [8]–[10]. Among these, Scharf's self-organizing fuzzy controller on robotics application [7], [9] can create and modify the rules which form the control strategy by referencing to an incremental performance matrix to adjust the rule matrix. Sugeno's fuzzy car [10], which can be trained to turn and park by itself, is another interesting example. He treated the consequence of a rule as a linear equation of the process state variables instead of a membership function. In this way, the problem is reduced to parameter estimation which can be done by optimizing a least-square performance index via a weighted linear regression method. Although these methods showed promising results, they are subjective and somewhat heuristic, and the choice of membership functions still depends on trial-and-error. Hence, bringing the learning abilities of neural networks to fuzzy logic systems may provide a more promising approach.

Neural networks have a large number of highly interconnected processing elements (nodes) that demonstrate the ability to learn and generalize from training patterns or data. They, like humans, can perform pattern-matching tasks, while traditional computer architecture, however, is inefficient at these tasks. On the contrary, the latter is faster at algorithmic computational tasks. Image/speech processing, robotics, nonmodel-based control, and decision making are all forms of pattern matched to a corresponding output. Neural networks, like fuzzy logic control/decision systems, are excellent at developing human-made systems that can perform the same type of information processing that our brain performs. There are many successful applications of neural networks in a variety of areas. Examples include Sejnowski/Rosenberg's NETtalk for text-to-speech conversion [11], Fukushima *et al.*'s "neocognitron" for visually recognizing hand-written Arabic numbers [12], Grossberg's vision processing network to synthesize coherently three-dimensional form, color, and brightness percepts [13], the control of cart–pole problem by Barto *et al.* [14] and Anderson [15], and the neural network for nonlinear self-tuning adaptive control [16].

Manuscript received February 1, 1991; revised June 22, 1991. This work was supported in part by the National Science Foundation under Grant CDR 8803017 to the Engineering Research Center for Intelligent Manufacturing Systems and a grant from the Ford Foundation. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

The authors are with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

IEEE Log Number 9103498.

Distributed representation and learning capabilities are two major features of neural networks. In distributed representation, a value is represented by a pattern of activity distributed over many computing elements, and each computing element is involved in representing many different values. So each computing element has a receptive field which is the set of all values which include all the patterns it represents. Therefore, each computing element corresponds to a fuzzy set, and its receptive field corresponds to the membership function. This coincidence brings the advantages of distributed representation in neural networks, which includes efficient hardware usage, easy generalization, and fault tolerance [17], into fuzzy logic systems. Learning ability is another strength in neural networks. Among the three classes of learning schemes [18], the unsupervised procedures, which construct internal models that capture regularities in their input vectors without receiving any additional information [19], are suitable to find clusters of data indicating the presence of fuzzy rules. The supervised procedures, which require a teacher to specify the desired output vector, and the reinforcement procedures, which only require a single scalar evaluation of the output, are good to adapt the rules or membership functions for the desired output in fuzzy logic systems. In [20], feedforward multilayer neural networks are trained through backpropagation (supervised learning) to act as membership functions for phonemes or syllables in acoustic cue detection, and fuzzy logic is applied on the output of neural networks for speech recognition. In [21], reinforcement/learning is used to adjust the consequence of fuzzy logic rules for the cart–pole balancing problem. Kosko proposed a system, called fuzzy cognitive maps, to integrate neural network and fuzzy logic, and the differential Hebbian learning law is used to learn the causal structure of the system [22]. In these approaches, either the membership functions or fuzzy logic rules are still chosen subjectively or the complete fuzzy control/decision problem was not solved.

In this paper, a general neural-network (connectionist) model is proposed for fuzzy logic control and decision systems. This connectionist model, in the form of feedforward multilayer net, combines the idea of fuzzy logic controller and neural-network structure and learning abilities into an integrated neural-network-based fuzzy logic control and decision system. A fuzzy logic control/decision network is constructed automatically by learning the training examples itself. In this connectionist structure, the input and output nodes represent the input states and output control/decision signals, respectively, and in the hidden layers there are nodes functioning as membership functions and rules. The learning algorithm for this network combines unsupervised learning and supervised gradient-descent learning procedures to build the rule nodes and train the membership functions. This hybrid learning algorithm performs superiorly to the purely supervised learning algorithm (e.g., backpropagation learning rule [23]) due to the *a priori* classification of training data through an overlapping receptive field before the supervised learning. The learning rate of the original backpropagation learning algorithm is limited by the fact that all layers of weights in the network are determined by the minimization of an error signal which is specified only as a function of the output, and a substantial fraction of the learning time is spent in the discovery of internal representation. This observation is also found in other reports [24], [25]. Also, the proposed connectionist model maintains the spirit of human thinking and reasoning as in fuzzy logic systems. This eliminates the disadvantage of a normal feedforward multilayer net which is difficult for an outside observer to understand or to modify. So, if necessary, experts' knowledge can be easily incorporated into the proposed structure. The connectionist structure also saves the rule-matching time of the inference engine in the traditional fuzzy logic system.

In Section II, the traditional fuzzy logic system is introduced and the proposed general connectionist model is then described. The hybrid learning algorithms are presented in Section III. Also, a specific structure from the general connectionist model is presented and its detailed learning algorithm is derived. In Section IV, two typical examples are simulated to demonstrate the fundamental applications of this model: fuzzy logic controller and fuzzy decision-making system. The issues of computational overhead and the convergence property of the proposed connectionist model is discussed in Section V. Conclusion and future research are summarized in the last section.

II. General Connectionist Fuzzy Logic Control/Decision System

We shall briefly introduce basic components in a traditional fuzzy logic system (for detailed discussion, please refer to [3]), and then propose our connectionist model.

A. Basic Structure of Fuzzy Logic System

Fig. 1 shows the basic structure of a traditional fuzzy logic system with a learning/adapting component. If the output of the defuzzifier feeds into the plant to act as a command, it functions as a fuzzy logic controller; otherwise, it functions as a fuzzy logic decision/diagnosis system. Before further examining the blocks in Fig. 1, we shall define *fuzzy sets and linguistic variables*. A fuzzy set F in a universe of discourse U is characterized by a membership function μ_F which takes values in the interval [0,1]; that is, $\mu_F : U \rightarrow [0,1]$. Thus, a fuzzy set F in U may be represented as a set of ordered pairs. Each pair consists of a generic element u and its grade of membership function; that is, $F = \{(u, \mu_F(u)) | u \in U\}$. u is called a *support value* if $\mu_F(u) > 0$. If U is a continuous universe and F is normal and convex (i.e., $\max_{u \in U} \mu_F(u) = 1$ and $\mu_F(\lambda u_1 + (1-\lambda)u_2) \geq \min(\mu_F(u_1), \mu_F(u_2))$, $u_1, u_2 \in U, \lambda \in [0,1]$), then F is a *fuzzy number*. A linguistic variable x in a universe of discourse U is characterized by $T(x) = \{T_x^1, T_x^2, \cdots, T_x^k\}$ and $M(x) = \{M_x^1, M_x^2, \cdots, M_x^k\}$, where $T(x)$ is the *term set* of x; that is, the set of names of linguistic values of x with each value T_x^i being a fuzzy number with membership function M_x^i defined on U. So $M(x)$ is a semantic rule for associating each value with its meaning. For example, if x indicates speed, then $T(x)$ may be {slow, medium, fast}. Following the above definition, the input vector X which includes the input state linguistic variables x_i's,

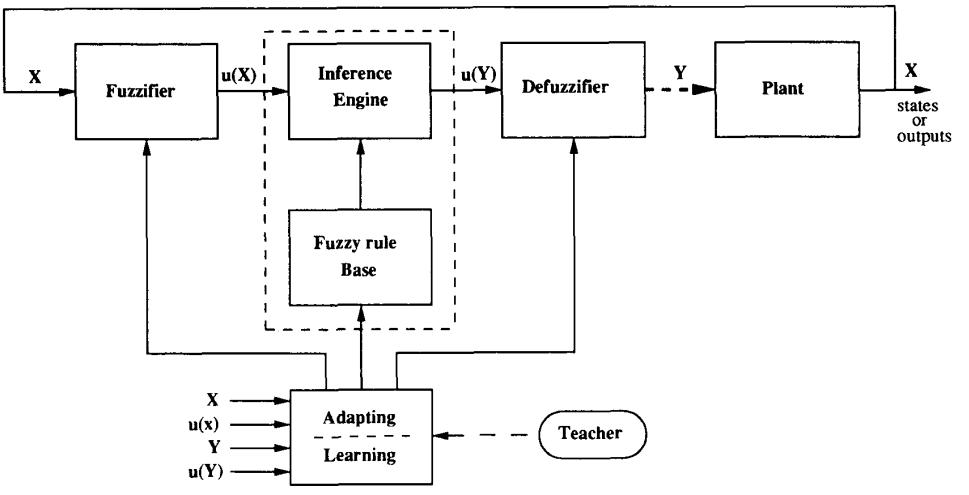


Fig. 1. General model of fuzzy logic controller and decision making (diagnosis) system.

and the output state vector Y which includes the output state linguistic variables y_i's in Fig. 1 can be defined as

$$X = \{(x_i, U_i, \{T_{x_i}^1, T_{x_i}^2, \cdots, T_{x_i}^{k_i}\}, \{M_{x_i}^1, M_{x_i}^2, \cdots, M_{x_i}^{k_i}\})|_{i=1\cdots n}\} \quad (1)$$

$$Y = \{(y_i, U_i', \{T_{y_i}^1, T_{y_i}^2, \cdots, T_{y_i}^{l_i}\}, \{M_{y_i}^1, M_{y_i}^2, \cdots, M_{y_i}^{l_i}\})|_{i=1\cdots m}\}. \quad (2)$$

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

The fuzzy rule base in Fig. 1 contains a set of fuzzy logic rules R. For a MIMO (multiinput and multioutput) system,

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

where the ith fuzzy logic rule is

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

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

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

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

IF the speed is TOO SLOW and the acceleration is DECELERATING,
THEN INCREASE POWER GREATLY.

The inference engine in Fig. 1 is to match the preconditions of rules in the fuzzy rule base with the input state linguistic terms and perform implication. For example, if there were two rules:

R1: IF x_1 is $T_{x_1}^1$ and x_2 is $T_{x_2}^1$, THEN y is T_y^1.
R2: IF x_1 is $T_{x_1}^2$ and x_2 is $T_{x_2}^2$, THEN y is T_y^2.

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

$$\alpha_1 = M_{x_1}^1(x_1) \wedge M_{x_2}^1(x_2) \quad (3)$$

where $\wedge$ is the *fuzzy AND* operation. The most commonly used fuzzy AND operations are intersection and algebraic product. Hence, α_i, $i = 1, 2$, becomes

$$\alpha_i = M_{x_1}^i(x_1) \wedge M_{x_2}^i(x_2) = \begin{cases} \min(M_{x_1}^i(x_1), M_{x_2}^i(x_2)) \\ \text{or} \\ M_{x_1}^i(x_1) M_{x_2}^i(x_2) \end{cases}. \quad (4)$$

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

$$\hat{M}_y^i(w) = \alpha_i \wedge M_y^i(w) = \begin{cases} \min(\alpha_i, M_y^i(w)) \\ \text{or} \\ \alpha_i M_y^i(w) \end{cases} \quad (5)$$

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

$$\hat{M}_y(w) = \hat{M}_y^1(w) \vee \hat{M}_y^2(w) \quad (6)$$

where $\vee$ is the *fuzzy OR* operation. The most commonly used fuzzy OR operations are union and bounded sum. Hence, the

output decision becomes

$$\hat{M}_y(w) = \hat{M}_y^1(w) \vee \hat{M}_y^2(w) = \begin{cases} \max(\hat{M}_y^1(w),\ \hat{M}_y^2(w)) \\ \quad \text{or} \\ \min(1,\ (\hat{M}_y^1(w) + \hat{M}_y^2(w)) \end{cases}. \tag{7}$$

Notice that the last result is a membership function curve. Before feeding the signal to the plant, we need a defuzzification process to get a crisp decision, and the defuzzifier block in Fig. 1 serves this purpose. Among the commonly used defuzzification strategies, the *center of area* method yields a superior result [26]. Let w_j be the support value at which the membership function, $\hat{M}_y^j(w)$, reaches the maximum value $\hat{M}_y^j(w)|_{w=w_j}$, then the defuzzification output is

$$y = \frac{\sum_j \hat{M}_y^j(w_j) w_j}{\sum_j \hat{M}_y^j(w_j)}. \tag{8}$$

The above describes the standard function operations in a traditional fuzzy logic control/decision system. There are some alternatives for fuzzy OR, fuzzy AND, and reasoning operations [3]. For example, Sugeno [10] proposed that the consequence of a rule is a function of input linguistic variables (in his application, he used linear combination). In this case, the defuzzification process is unnecessary, and there are no exact membership functions for output linguistic variables. Currently, most system designers choose their membership functions empirically and construct the fuzzy logical rules from experts. Bringing learning abilities to the fuzzy logic system is an important issue. The learning/adapting block in Fig. 1 represents this function. This learning/adapting block finds suitable fuzzy logic rules and adapts the fuzzifier and the defuzzifier to find the proper shapes and overlaps of membership functions by learning the desired output or according to an external reinforcement signal.

Nowadays, the function blocks in Fig. 1 are usually implemented by software or firmware in microprocessor systems. In such systems, table lookup and serial function computations are the usual schemes to perform the fuzzification and the defuzzification processes. The fuzzy rules are stored in rule-set memories, and the rule-matching process is performed by the inference engine like the general expert system architecture. For smaller systems, control table lookup is another technique to reason fuzzy logic rules. Recently, some fuzzy logic chips were designed [27], [28] to speed up the whole process. These include the fuzzy inference chip which employs MAX and MIN operations to reduce rules matching and rule composition time, the membership function generator, and the defuzzifier chip. To provide learning/adapting capability in the traditional fuzzy logic control/decision structure, extra learning/adapting architecture and connections must be added. In the next subsection, a connectionist model is proposed. This neural-network-based architecture eliminates the rule-matching process and distributively stores the control/decision knowledge in the connections and link weights. This connectionist architecture is also suitable for VLSI implementation. The control/decision signal can be pumped out immediately right after the input data are fed into it. More importantly, the connectionist architecture is a natural structure to perform neural-network-based learning. A hybrid learning algorithm is developed in Section III to learn and adapt membership functions and fuzzy logic rules. This learning function is also distributed in the proposed connectionist structure.

B. Connectionist Fuzzy Logic Control and Decision System

Fig. 2 shows the proposed connectionist fuzzy logic control/decision system. The system has a total of five layers. Nodes at layer one are input nodes (*linguistic nodes*) which represent input linguistic variables. Layer five is the output layer. We have two linguistic nodes for each output variable. One is for training data (desired output) to feed into this net, and the other is for decision signal (actual output) to be pumped out of this net. Nodes at layers two and four are *term nodes* which act as membership functions to represent the terms of the respective linguistic variable. Actually, a layer-two node can be either a single node that performs a simple membership function (e.g., a triangle-shaped or a bell-shaped function) or composed of multilayer nodes (a subneural net) to perform a complex membership function (e.g., in an acoustic cue detector [20]). So the total number of layers in this connectionist model can be more than five. Each node at layer three is a rule node which represents one fuzzy rule. Thus, all the layer-three nodes form a fuzzy rule base. Links at layers three and four function as a *connectionist inference engine* [29], [30], which avoids the rule-matching process. Layer-three links define the preconditions of the rule nodes, and layer-four links define the consequences of the rule nodes. Therefore, for each rule node, there is at most one link (maybe none) from some term node of a linguistic node. This is true both for precondition links (links at layer three) and consequence links (links at layer four). The links at layers two and five are fully connected between linguistic nodes and their corresponding term nodes. The arrow on the link indicates the normal signal flow direction when this network is in use after it has been built and trained. We shall later indicate the signal propagation, layer by layer, according to the arrow direction. Signals may flow in the reverse direction in the learning process as discussed in Section III.

With this five-layered structure of the proposed connectionist model, we shall then define the basic functions of a node. A typical network consists of a unit which has some finite fan-in of connections represented by weight values from other units and fan-out of connections to other units (see Fig. 3). Associated with the fan-in of a unit is an integration function f which serves to combine information, activation, or evidence from other nodes. This function provides the net input for this node

$$\text{net-input} = f(u_1^k, u_2^k, \cdots, u_p^k; w_1^k, w_2^k, \cdots, w_p^k) \tag{9}$$

where the superscript indicates the layer number. This notation will be also used in the following equations. A second action of each node is to output an activation value as a function of its net-input,

$$\text{output} = o_i^k = a(f) \tag{10}$$

where $a(\cdot)$ denotes the activation function. For example (in a

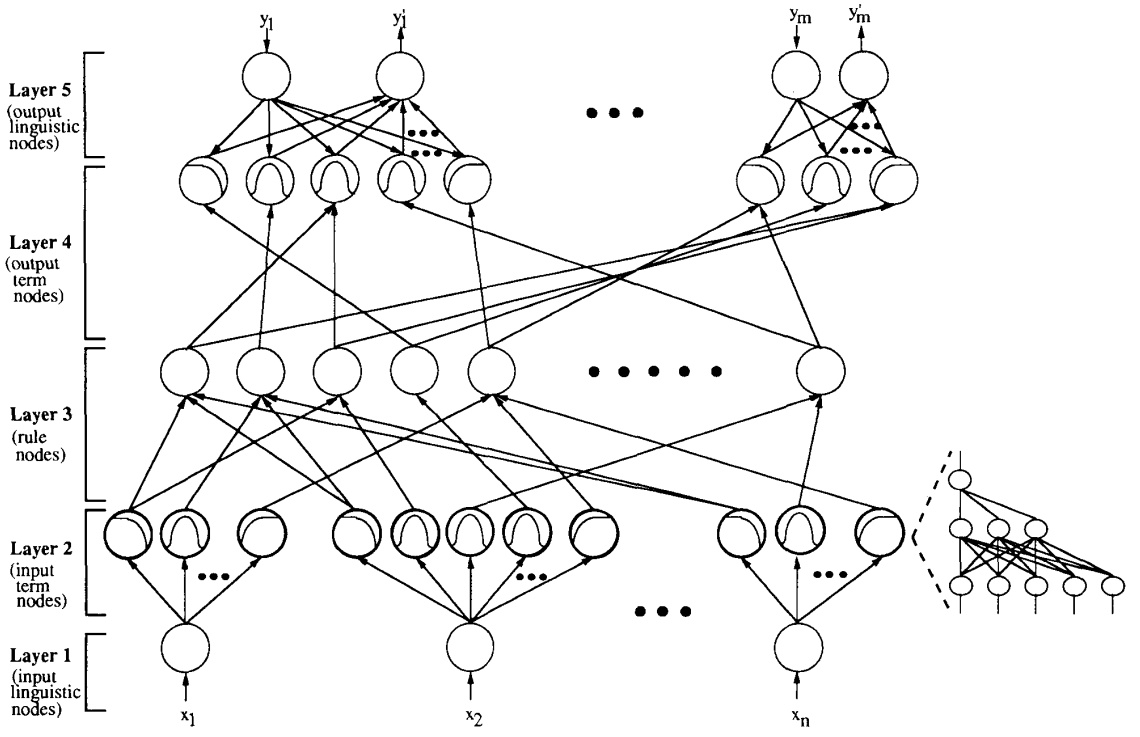


Fig. 2. Connectionist fuzzy logic control/decision system.

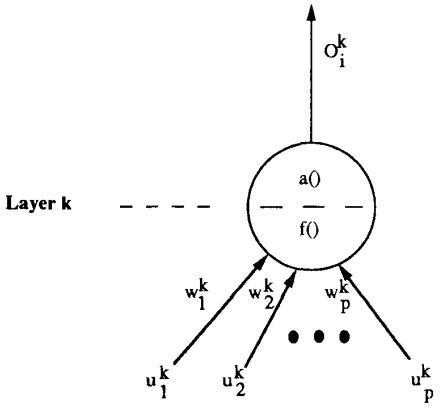


Fig. 3. Basic structure of a node in a neural network.

standard form),

$$f = \sum_{i=1}^{p} w_i^k u_i^k \text{ and } a = \frac{1}{1+e^{-f}}. \quad (11)$$

We shall next describe the functions of the nodes in each of the five layers of the proposed connectionist model.

Layer 1: The nodes in this layer just transmit input values to the next layer directly. That is,

$$f = u_i^1 \text{ and } a = f. \quad (12)$$

From the above equation, the link weight at layer one (w_i^1) is unity.

Layer 2: If we use a single node to perform a simple membership function, then the output function of this node should be this membership function. For example, for a bell-shaped function,

$$f = M_{x_i}^j(m_{ij}, \sigma_{ij}) = -\frac{(u_i^2 - m_{ij})^2}{\sigma_{ij}^2} \text{ and } a = e^f \quad (13)$$

where m_{ij} and σ_{ij} are, respectively, the center (or mean) and the width (or variance) of the bell-shaped function of the jth term of the ith input linguistic variable x_i. Hence, the link weight at layer two (w_{ij}^2) can be interpreted as m_{ij}. If we use a set of nodes to perform a membership function, then the function of each node can be just in the standard form [(11)], and the whole subnet is trained off-line to perform the desired membership function by a standard learning algorithm (e.g., backpropagation [24]).

Layer 3: The links in this layer are used to perform precondition matching of fuzzy logic rules. Hence, the rule nodes should perform the fuzzy AND operation,

$$f = \min(u_1^3, u_2^3, \cdots, u_p^3) \text{ and } a = f. \quad (14)$$

The link weight in layer three (w_i^3) is then unity.

Layer 4: The nodes in this layer have two operation modes: *down–up* transmission and *up–down* transmission modes. In the *down–up* transmission mode, the links at layer four should

perform the fuzzy OR operation to integrate the fired rules which have the same consequence,

$$f = \sum_{i=1}^{p} u_i^4 \text{ and } a = \min(1, f). \tag{15}$$

Hence the link weight $w_i^4 = 1$. In the up–down transmission mode, the nodes in this layer and the links in layer five function exactly the same as those in layer two except that only a single node is used to perform a membership function for output linguistic variables.

Layer 5: There are two kinds of nodes in this layer. The first kind of node performs the up–down transmission for the training data to feed into the network. For this kind of node,

$$f = y_i \text{ and } a = f. \tag{16}$$

The second kind of node performs the down–up transmission for the decision signal output. These nodes and the layer-five links attached to them act as the defuzzifier. If m_{ij}^5's and σ_{ij}^5's are the centers and the widths of membership functions, respectively, then the following functions can be used to simulate the *center of area* defuzzification method [3]:

$$f = \sum w_{ij}^5 u_i^5 = \sum (m_{ij}\sigma_{ij}) u_i^5 \text{ and } a = \frac{f}{\sum \sigma_{ij} u_i^5}. \tag{17}$$

Here the link weight at layer five (w_{ij}^5) is $m_{ij}\sigma_{ij}$.

Based on the above connectionist structure, a hybrid learning algorithm is developed in Section III. This two-stage learning algorithm will determine optimal centers (m_{ij}'s) and widths (σ_{ij}'s) of term nodes in layers two and four. Also, it will learn fuzzy logic rules by deciding the existence and connection types of the links at layers three and four; that is, the precondition links and consequence links of the rule nodes.

III. Hybrid Learning Algorithm

In this section, we shall present a two-phase learning scheme for the proposed connectionist model. In phase one, a self-organized learning scheme is used to locate initial membership functions and to find the presence of rules. In phase two, a supervised learning scheme is used to optimally adjust the membership functions for desired outputs. To initiate the learning scheme, training data and the desired or guessed coarse of fuzzy partition (i.e., the size of the term set of each input/output linguistic variable) must be provided from the outside world.

Before this network is trained, an initial form of the network is first constructed. Then, during the learning process, some nodes and links of this initial network are deleted or combined to form the final structure of the network. In its initial form (see Fig. 2), there are $\prod_i |T(x_i)|$ rule nodes with the inputs of each rule node coming from one possible combination of the terms of input linguistic variables under the constraint that only one term in a term set can be a rule node's input. Here $|T(x_i)|$ denotes the number of terms of x_i (i.e., the number of fuzzy partitions of input state linguistic variable x_i). So the state space is initially divided into $|T(x_1)| \times |T(x_2)| \times \cdots \times |T(x_n)|$ linguistically defined nodes (or fuzzy cells) which represent the preconditions of fuzzy rules. Also, initially, the links between the rule nodes and the output term nodes are fully connected, meaning that the consequences of the rule nodes are not yet decided. Only a suitable term in each output linguistic variable's term set will be chosen after the learning process.

A. Self-Organized Learning Phase

The problem for the self-organized learning can be stated as: Given the training input data $x_i(t)$, $i = 1, \cdots, n$ [see (1)], the desired output value $y_i(t)$, $i = 1, \cdots, m$ [see (2)], the fuzzy partitions $|T(x)|$ and $|T(y)|$, and the desired shapes of membership functions, we want to locate the membership functions and find the fuzzy logic rules. In this phase, the network works in a two-sided manner; that is, the nodes and links at layer four are in the up–down transmission mode so that the training input and output data are fed into this network from both sides.

First, the centers (or means) and the widths (or variances) of the membership functions are determined by self-organized learning techniques analogous to statistical clustering. This serves to allocate network resources efficiently by placing the domains of membership functions covering only those regions of the input/output space where data are present. Kohonen's feature-maps algorithm [31] is adapted here to find the center m_i of the membership function:

$$\|x(t) - m_{\text{closest}}(t)\| = \min_{1 \le i \le k} \{\|x(t) - m_i(t)\|\} \tag{18}$$

$$m_{\text{closest}}(t+1) = m_{\text{closest}}(t) + \alpha(t)[x(t) - m_{\text{closest}}(t)] \tag{19}$$

$$m_i(t+1) = m_i(t) \text{ for } m_i \neq m_{\text{closest}} \tag{20}$$

where $\alpha(t)$ is a monotonically decreasing scalar learning rate, and $k = |T(x)|$. This adaptive formulation runs independently for each input and output linguistic variable. The determination of which of the m_i's is m_{closest} can be accomplished in constant time via a winner-take-all circuit.

Once the centers of membership functions are found, their widths can be determined by the *N-nearest-neighbors* heuristic by minimizing the following objective function with respect to the widths σ_i's:

$$E = \frac{1}{2} \sum_{i=1}^{N} \left[\sum_{j \in N_{\text{nearest}}} \left(\frac{m_i - m_j}{\sigma_i} \right)^2 - r \right]^2 \tag{21}$$

where r is an overlap parameter. Since our second learning phase will optimally adjust the centers and the widths of the membership functions, the widths can be simply determined by the first-nearest-neighbor heuristic at this stage as

$$\sigma_i = \frac{|m_i - m_{\text{closest}}|}{r}. \tag{22}$$

After the parameters of the membership functions have been found, the signals from both external sides can reach the output points of term nodes at layer two and layer four (see Fig. 2). Furthermore, the outputs of term nodes at layer two can be transmitted to rule nodes through the initial architecture of layer-three links. So we can get the firing strength of each

rule node. Based on these rule firing strengths (denoted as $o_i^3(t)$'s) and the outputs of term nodes at layer four (denoted as $o_j^4(t)$'s), we want to decide the correct consequence links (layer four links) of each rule node to find the existing fuzzy logic rule by competitive learning algorithms [19], [32], [33]. As stated in the last section, the links at layer four are initially fully connected. We denote the weight of the link between the ith rule node and the jth output term node as w_{ij}. The following competitive learning law is used to update these weights for each training data set,

$$\dot{w}_{ij}(t) = o_j^4(-w_{ij} + o_i^3). \tag{23}$$

Here o_j^4 serves as a win–loss index of the jth term node at layer four. The theme of this law is that *learn if win*. In the extreme case, if o_j^4 is a 0–1 threshold function, then the above law says *learn only if win*. The convergence proof of this law can be found in the Appendix.

After the competitive learning through the whole training data set, the link weights at layer four represent the strength of the existence of the corresponding rule consequence. Among the links which connect a rule node and the term nodes of an output linguistic node, at most one link with maximum weight is chosen and the others are deleted. Hence, only one term in an output linguistic variable's term set can become one of the consequences of a fuzzy logic rule. If all the link weights between a rule node and the term nodes of an output linguistic node are very small, then all the corresponding links are deleted, meaning that this rule node has little or no relation to this output linguistic variable. If all the links between a rule node and the layer-four nodes are deleted, then this rule node can be eliminated since it does not affect the outputs.

After the consequences of rule nodes are determined, the rule combination is performed to reduce the number of rules. The criteria for a set of rule nodes to be combined into a single rule node are: 1) they have exactly the same consequences, 2) some preconditions are common to all the rule nodes in this set, 3) the union of other preconditions of these rule nodes composes the whole term set of some input linguistic variables. If a set of nodes meets these criteria, a new rule node with only the common preconditions can replace this set of rule nodes. An example is illustrated in Fig. 4.

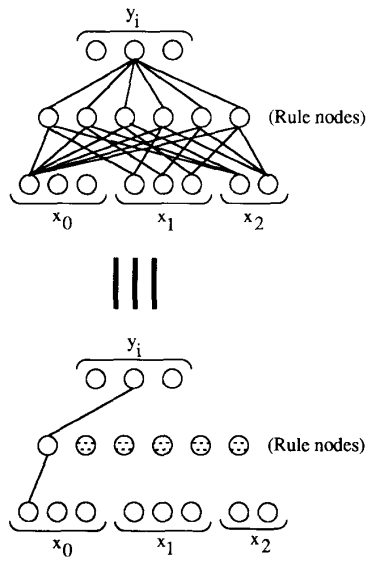


Fig. 4. Example of combination of rule nodes.

B. Supervised Learning Phase

After the fuzzy logic rules have been found, the whole network structure is established, and the network then enters the second learning phase to adjust the parameters of the membership functions optimally. The problem for the supervised learning can be stated as: Given the training input data $x_i(t)$, $i = 1, \cdots, n$ [see (1)], the desired output value $y_i(t)$, $i = 1, \cdots, m$ [see (2)], the fuzzy partitions $|T(x)|$ and $|T(y)|$, and the fuzzy logic rules, adjust the parameters of the membership functions optimally. These fuzzy logic rules were learned in the first-phase learning or, in some application domains, they can be given by experts. The latter is feasible since the expert just needs to give such kind of rule: "If temperature is too high, then turn it down greatly." In the second-phase learning, the network works in the feedforward manner; that is, the nodes and the links at layer four are in the down–up transmission mode. The idea of backpropagation [18], [24] is used for this supervised learning. The goal is to minimize the error function

$$E = \frac{1}{2}(y(t) - \hat{y}(t))^2 \tag{24}$$

where $y(t)$ is the desired output, and $\hat{y}(t)$ is the current output. For each training data set, starting at the input nodes, a forward pass is used to compute the activity levels of all the nodes in the network. Then starting at the output nodes, a backward pass is used to compute $\frac{\partial E}{\partial y}$ for all the hidden nodes. Assuming that w is the adjustable parameter in a node (e.g., the center of a membership function), the general learning rule used is

$$\Delta w \propto -\frac{\partial E}{\partial w} \tag{25}$$

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

where η is the learning rate, and

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial(\text{net-input})}\frac{\partial(\text{net-input})}{\partial w} = \frac{\partial E}{\partial f}\frac{\partial f}{\partial w} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial f}\frac{\partial f}{\partial w}. \tag{27}$$

To show the learning rule, we shall show the computations of $\frac{\partial E}{\partial y}$, layer by layer, starting at the output nodes, and we will use the bell-shaped membership functions with centers m_i's and widths σ_i's as the adjustable parameters for these computations.

Layer 5: Using (27) and (17), the adaptive rule of the center m_i is derived as:

$$\frac{\partial E}{\partial m_i} = \frac{\partial E}{\partial a}\frac{\partial a}{\partial f^5}\frac{\partial f^5}{\partial m_i} = -[y(t) - \hat{y}(t)]\frac{\sigma_i u_i}{\sum \sigma_i u_i}. \tag{28}$$

Hence, the center parameter is updated by

$$m_i(t+1) = m_i(t) + \eta[y(t) - \hat{y}(t)]\frac{\sigma_i u_i}{\sum \sigma_i u_i}. \quad (29)$$

Similarly, using (27) and (17), the adaptive rule of the width σ_i is derived as

$$\begin{aligned}\frac{\partial E}{\partial \sigma_i} &= \frac{\partial E}{\partial a}\frac{\partial a}{\partial f^5}\frac{\partial f^5}{\partial \sigma_i} \\ &= -[y(t) - \hat{y}(t)] \\ &\quad \cdot \frac{m_i u_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i)u_i}{(\sum \sigma_i u_i)^2}. \end{aligned} \quad (30)$$

Hence, the width parameter is updated by

$$\begin{aligned}\sigma_i(t+1) =& \sigma_i(t) + \eta[y(t) - \hat{y}(t)] \\ &\cdot \frac{m_i u_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i)u_i}{(\sum \sigma_i u_i)^2}. \end{aligned} \quad (31)$$

The error to be propagated to the preceding layer is

$$\delta^5 = \frac{-\partial E}{\partial f^5} = \frac{-\partial E}{\partial a}\frac{\partial a}{\partial f^5} = y(t) - \hat{y}(t). \quad (32)$$

Layer 4: In the down–up mode, there is no parameter to be adjusted in this layer. Only the error signals (δ_i^4's) need to be computed and propagated. The error signal δ_i^4 is derived as in the following:

$$\begin{aligned}-\delta_i^4 &= \frac{\partial E}{\partial f_i} = \frac{\partial E}{\partial a_i}\frac{\partial a_i}{\partial f_i} \\ &= \frac{\partial E}{\partial (\text{net-input})^5}\frac{\partial (\text{net-input})^5}{\partial a_i} \end{aligned} \quad (33)$$

where [from (17)]

$$\frac{\partial (\text{net-input})^5}{\partial a_i} = \frac{\partial f^5}{\partial u_i^5} = \frac{m_i \sigma_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i)\sigma_i}{(\sum \sigma_i u_i)^2} \quad (34)$$

and from (32),

$$\frac{\partial E}{\partial (\text{net-input})^5} = \frac{\partial E}{\partial f^5} = -\delta^5 = -[y(t) - \hat{y}(t)]. \quad (35)$$

Hence, the error signal is

$$\delta_i^4(t) = [y(t) - \hat{y}(t)]\frac{m_i \sigma_i(\sum \sigma_i u_i) - (\sum m_i \sigma_i u_i)\sigma_i}{(\sum \sigma_i u_i)^2}. \quad (36)$$

In the multiple-output case, the computations in layers five and four are exactly the same as the above and proceed independently for each output linguistic variable.

Layer 3: As in layer four, only the error signals need to be computed. According to (15), this error signal can be derived as

$$\begin{aligned}-\delta_i^3 &= \frac{\partial E}{\partial f_i} = \frac{\partial E}{\partial a_i}\frac{\partial a_i}{\partial f_i} = \frac{\partial E}{\partial (\text{net-input})^4}\frac{\partial (\text{net-input})^4}{\partial a_i} \\ &= -\delta_i^4 \frac{\partial f^4}{\partial u_i^4} = -\delta_i^4. \end{aligned} \quad (37)$$

Hence, the error signal is $\delta_i^3 = \delta_i^4$. If there are multiple outputs, then the error signal becomes $\delta_i^3 = \sum_k \delta_k^4$, where the summation is performed over the consequences of a rule node; that is, the error of a rule node is the summation of the errors of its consequences.

Layer 2: Using (27) and (13), the adaptive rule of m_{ij} is derived as in the following:

$$\frac{\partial E}{\partial m_{ij}} = \frac{\partial E}{\partial a_i}\frac{\partial a_i}{\partial f_i}\frac{\partial f_i}{\partial m_{ij}} = \frac{\partial E}{\partial a_i} e^{f_i}\frac{2(u_i - m_{ij})}{\sigma_{ij}^2} \quad (38)$$

where [from (37)]

$$\frac{\partial E}{\partial a_i} = \sum_k \frac{\partial E}{\partial (\text{net-input})^k}\frac{\partial (\text{net-input})^k}{\partial a_i} \quad (39)$$

$$\frac{\partial E}{\partial (\text{net-input})^k} = \frac{\partial E}{\partial f_k^3} = \delta_k^3 \quad (40)$$

and from (14),

$$\begin{aligned}\frac{\partial (\text{net-input})^k}{\partial a_i} &= \frac{\partial f^3}{\partial u_i^3} \\ &= \begin{cases} 1, & \text{if } u_i^3 = \min(\text{inputs of rule node } k) \\ 0, & \text{otherwise} \end{cases}. \end{aligned} \quad (41)$$

Hence,

$$\frac{\partial E}{\partial a_i} = \sum_k q_k \quad (42)$$

where the summation is performed over the rule nodes that a_i feeds into, and

$$q_k = \begin{cases} \delta_k^3, & \text{if } a_i \text{ is minimum in } k\text{th rule node's inputs} \\ 0, & \text{otherwise} \end{cases}. \quad (43)$$

So the adaptive rule of m_{ij} is

$$m_{ij}(t+1) = m_{ij}(t) - \eta\frac{\partial E}{\partial a_i} e^{f_i}\frac{2(u_i - m_{ij})}{\sigma_{ij}^2}. \quad (44)$$

Similarly, using (27), (13), and (39)–(43), the adaptive rule of σ_{ij} is derived as

$$\frac{\partial E}{\partial \sigma_{ij}} = \frac{\partial E}{\partial a_i}\frac{\partial a_i}{\partial f_i}\frac{\partial f_i}{\partial \sigma_{ij}} = \frac{\partial E}{\partial a_i} e^{f_i}\frac{2(u_i - m_{ij})^2}{\sigma_{ij}^3}. \quad (45)$$

Hence, the adaptive rule of σ_{ij} becomes

$$\sigma_{ij}(t+1) = \sigma_{ij}(t) - \eta\frac{\partial E}{\partial a_i} e^{f_i}\frac{2(u_i - m_{ij})^2}{\sigma_{ij}^3}. \quad (46)$$

The whole learning procedure is summarized by the flow chart in Fig. 5. The convergence speed of the backpropagation is found superior to the normal backpropagation scheme since the self-organized learning process in phase one has done much of the learning work in advance. Finally, it should be noted that this backpropagation algorithm can be easily extended to train the membership function which is implemented by a subneural net instead of a single term node at layer two, since, from the above analysis, the error signal can be propagated to the output node of the subneural net. Then, by using a similar backpropagation rule in this subneural net, the parameters in this subneural net can be adjusted.

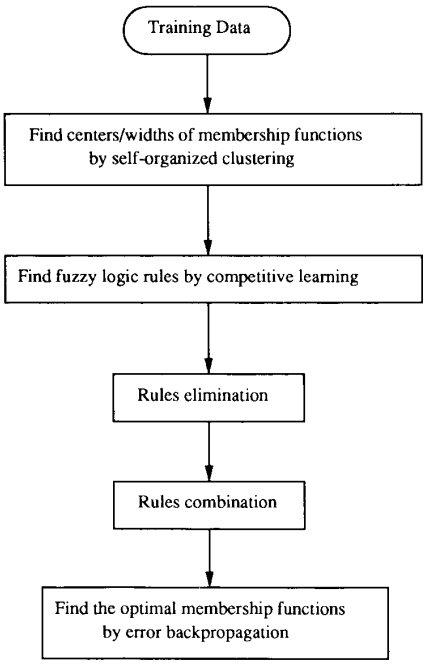


Fig. 5. Flow chart of hyprid learning procedure.

IV. Illustrative Examples

A general purpose simulator for the proposed connectionist model for fuzzy logic control/decision systems has been written in "C" language and runs on a Sun SPARCstation. Using this simulator, two typical examples are presented in this section to show the fundamental applications of the proposed model. The first example is to control an unmanned vehicle by learning the driving technique of a skilled driver, and the second example is to build a decision-making system for manufacturing scheduling.

A. Example 1: Fuzzy Control of Unmanned Vehicle

In this example, the proposed connectionist model for a fuzzy logic controller was used to simulate the control of the fuzzy car conceived by Sugeno [7], [10]. The car has the ability to learn from examples to move automatically along a track with rectangular turns. The input linguistic variables are x_0, x_1, x_2 which represent the distances of the car from the boundaries of the track at a corner and the current steering angle (see Fig. 6). The output linguistic variable y is the next steering angle. The training data are obtained in the process when an operator guides the fuzzy car along the track as shown in Fig. 9. Fig. 7 shows the learned membership functions of x_0, x_1, x_2, and y after the phase-one (with an overlap parameter $r = 2.0$) and the phase-two learnings. Table I shows the learned fuzzy logic rules; for example, rule 0 is interpreted as

IF x_0 is term 0 and x_1 is term 1 and x_2 is term 0,
THEN y is term 6.

These fuzzy logic rules also indicate the hidden-layered structure of the designed neural network as discussed in Subsection II-B. Fig. 8 shows the curve of mean error with respect to the number of epochs, and the curve of mean iteration number with respect to the number of epochs. Here the learning rate is 0.15 and the error tolerance is 0.01. From

TABLE I
The Learned Fuzzy Logic Rules in Example 1

Learned Fuzzy Logic rules									
	Preconditions			Consequence		Preconditions			Consequence
Rule	x_0	x_1	x_2	y	Rule	x_0	x_1	x_2	y
0	0	1	0	6	31	2	3	2	12
1	1	1	0	6	32	1	4	2	12
2	2	1	0	5	33	2	4	2	12
3	0	2	0	3	34	0	0	3	12
4	1	2	0	2	35	1	0	3	12
5	2	2	0	1	36	2	0	3	12
6	0	3	0	2	37	0	1	3	12
7	1	3	0	2	38	1	1	3	12
8	2	3	0	0	39	2	1	3	13
9	0	0	1	9	40	0	2	3	13
10	1	0	1	9	41	1	2	3	13
11	0	1	1	7	42	2	2	3	13
12	1	1	1	7	43	0	3	3	13
13	2	1	1	7	44	1	3	3	13
14	0	2	1	6	45	2	3	3	13
15	1	2	1	4	46	1	4	3	12
16	2	2	1	2	47	2	4	3	12
17	0	3	1	2	48	0	0	4	12
18	1	3	1	2	49	1	0	4	12
19	2	3	1	2	50	2	0	4	12
20	0	0	2	11	51	0	1	4	13
21	1	0	2	10	52	1	1	4	13
22	2	0	2	12	53	2	1	4	13
23	0	1	2	10	54	0	2	4	14
24	1	1	2	8	55	1	2	4	13
25	2	1	2	13	56	2	2	4	13
26	0	2	2	7	57	0	3	4	13
27	1	2	2	7	58	1	3	4	13
28	2	2	2	13	59	2	3	4	13
29	0	3	2	13	60	1	4	4	13
30	1	3	2	12	61	2	4	4	13

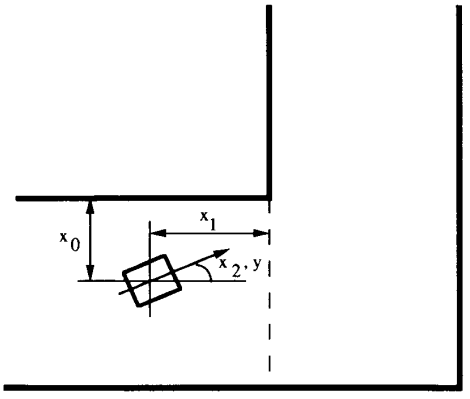


Fig. 6. The state variables of the fuzzy car in Example 1.

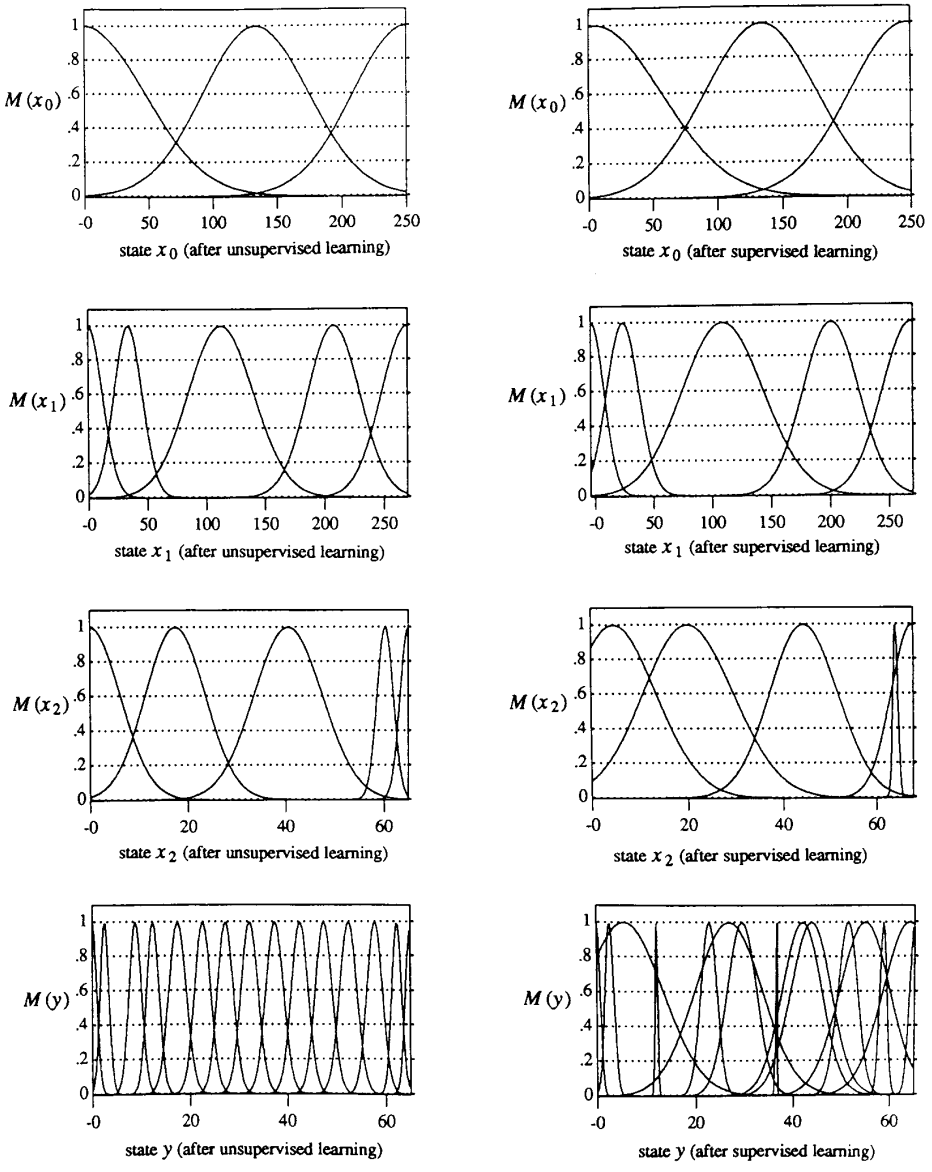


Fig. 7. Learned membership functions in Example 1

the second curve, we can find that the average iteration number for a single training point to converge is rather small from the beginning, meaning that the phase-one learning process has done much work already. After the whole connectionist fuzzy logic controller is established, it is used to control the car. We keep the speed of the car constant, and assume there are sensors on the car to measure the state values x_0, x_1, and x_2 which are fed into the controller to derive the next steering angle. The simulated result is shown in Fig. 9. The solid curve is the training path and the dotted curve is the path that the fuzzy car runs under the control of the proposed connectionist fuzzy logic controller. We found these paths coincide closely. We simulated this example several times with different initial steering angles, and the results we obtained were nearly the same.

B. Example 2: Fuzzy Logic Decision-Making System for Manufacturing Scheduling

In our second example, the proposed connectionist model for a fuzzy logic decision-making system was used to decide a proper scheduling strategy based on some chosen attributes in manufacturing industry. Fig. 10 shows a typical scheduling environment for a complete job shop [34], [35]. The arrows on this graph show the possible processing routings. For

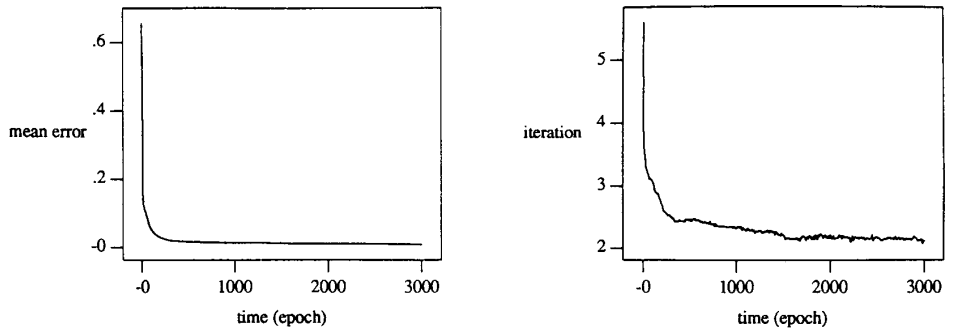


Fig. 8. Learning curves: mean error and mean iteration versus time (epoch) in Example 1.

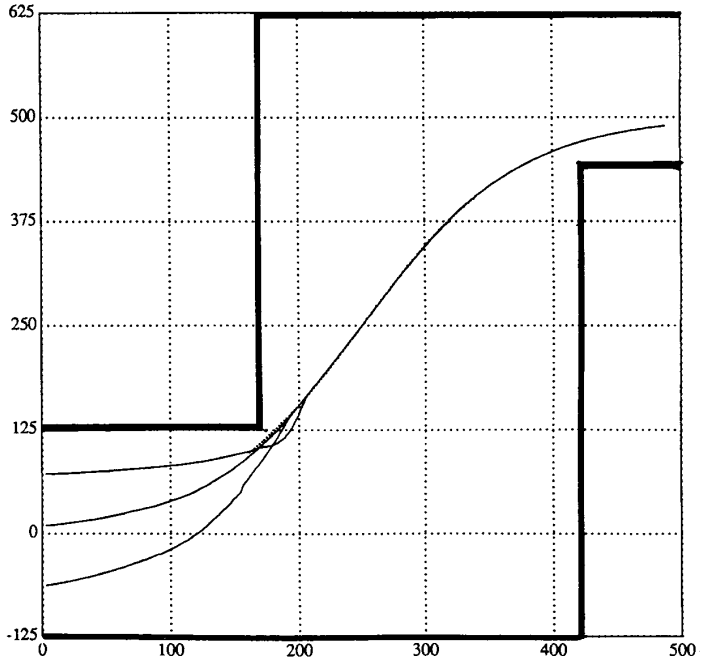
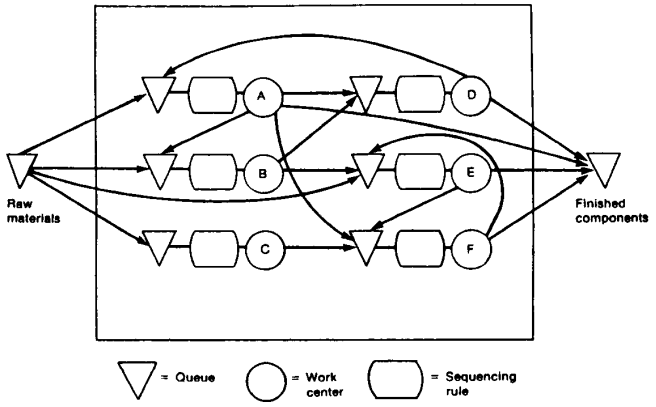


Fig. 9. Simulation result of the fuzzy car running under the control of the proposed connectionist controller.

example, after processing at machine center A, the job may be sent to machine center B, D, or F for further processing. A sequencing or dispatching rule is depicted in Fig. 10 between each queue (buffer) and its associated machine center. There is a dispatching rule which exists for choosing the next job in the queue for processing. Our problem is to determine which sequencing rule will achieve good performance based on some scheduling criterion. Among various sequencing rules [34], [35], three typical ones are chosen to be our candidates: 1) Least setup (LSU) rule which selects the job that minimizes the changeover time on the machine; 2) Earliest due day (EDD) rule which schedules the jobs in due day sequence; 3) Shortest processing time (SPT) rule which selects the next job for processing that has the shortest processing time at current operation. The selection of a proper scheduling rule will base on some chosen attributes, for example, the queue size, the system utilization, and the flow time (the time that a job spends in a system). The input linguistic variables $x_0, \cdots, x_5$ represent these attributes, and the output linguistic variables y_0, y_1, and y_2 represent the above three possible scheduling strategies, respectively. Their definitions are shown in Fig. 10. 200 training data sets were used in this simulation. After the phase-one (with an overlap parameter $r = 2.0$) and the phase-two learning processes, we obtained the membership functions of the linguistic variables as shown in Fig. 11. Table II indicates the learned fuzzy logic rules. For example, rule 1 says that if x_0 is term 1, then use y_2 (SPT) scheduling method. These fuzzy logic rules also indicate the hidden-layered structure of the designed neural network as discussed in Subsection II-B. Fig. 12 shows the learning curves in the



x_0 : Total buffer size

x_1 : Overall system utilization

x_2 : Contention factor (the average number of alternative machines available)

x_3 : Blocking status

x_4 : Variability in machine workload (measured by the ratio of the standard deviation of individual utilization to the average machine utilization)

x_5 : Flow allowance factor

y_0 : Least setup (LSU) rule : pick the job that minimizes the changeover time on the machine.

y_1 : Earliest due date (EDD) rule : schedule the jobs in due date sequence.

y_2 : Shortest processing time (SPT) rule : select the next job for processing that has the shortest processing time at the current operation.

Fig. 10. The state variables of the manufacturing scheduling problem in Example 2.

phase-two learning process. Here the learning rate is 0.15, and the error tolerance is 0.05. These curves appear to have a little oscillation at the end of the phase-two learning. This situation can be improved if we increase the numbers of fuzzy partitions of y_0, y_1, and y_2. The results are good enough for a decision-making system since it is required to determine exactly one method among the three candidates.

V. Issues on Computational Overhead and Convergence Property

The hardware requirement (i.e., the number of network elements and connections) is highly dependent on the number of input and output state variables in the control or decision-making environment and their corresponding number of fuzzy partitions (linguistic terms), which is usually less than 5 per state variable. These decide the number of neurons in all the layers except layer 3. The number of neurons in the third layer is bounded above by $\prod_i |T(x_i)|$, which is 75 in Example 1 and 192 in Example 2; however, the exact number will be decided after the first-phase learning. Fortunately, this final number is always much smaller than the upper bound in the minimal-node representation of fuzzy logic rules through the rule elimination and combination process at the end of the first-phase learning. For example, after the rule elimination and combination process, Example 1 requires only 61 neurons while Example 2 requires 34 neurons in their third layers.

Regarding the learning time of the proposed hybrid learning algorithm, the bottleneck is in the second-phase learning—backpropagation. In the first-phase learning, the computational time of Kohonen's feature-maps algorithm and the competitive learning law is measured in seconds in a SPARCstation (i.e., SPARC seconds), but the convergence time of the backpropagation in the second-phase learning is measured in SPARC minutes. However, this speed is still much faster than the standard backpropagation on normal, equivalent-size feedforward network, of which the typical convergence time is measured in SPARC hours or even days. Basically, three reasons can be accounted for the fast learning time of the hybrid learning algorithm in the proposed network architecture. First, although a large number of neurons may exist in the third layer, only the parameters in the second and fourth layers need to be learned. So even though it

TABLE II
THE LEARNED FUZZY LOGIC RULES IN EXAMPLE 2

Learned Fuzzy Logic rules									
	Preconditions						Consequence		
Rule	x_0	x_1	x_2	x_3	x_4	x_5	y_0	y_1	y_2
0	0	0	0	0	0	0	0	-	0
1	1	-	-	-	-	-	0	0	1
2	0	1	-	-	-	-	1	0	0
3	0	0	1	0	0	0	1	0	0
4	0	0	2	-	0	0	0	0	1
5	0	0	0	1	-	-	1	0	0
6	0	0	1	1	-	-	1	0	0
7	0	0	0	0	1	0	0	1	0
8	0	1	0	0	1	0	-	0	0
9	0	0	1	0	1	0	0	0	1
10	0	0	2	-	1	0	1	0	0
11	0	0	0	0	-	1	0	0	0
12	0	0	1	0	0	1	0	0	0
13	0	0	2	-	0	1	0	0	0
14	0	1	0	0	1	1	-	0	0
15	0	0	1	0	1	1	0	0	0
16	0	0	2	-	1	1	0	0	1
17	0	0	0	1	1	1	-	0	0
18	0	0	1	1	1	1	-	0	-
19	0	0	0	0	0	2	0	0	-
20	0	0	1	0	0	2	-	0	0
21	0	0	2	-	0	2	1	0	0
22	0	0	0	0	1	2	0	0	0
23	0	0	1	0	1	2	0	0	0
24	0	0	2	-	1	2	0	0	-
25	0	0	0	0	0	3	0	1	0
26	0	0	1	0	0	3	0	0	0
27	0	0	2	-	0	3	0	0	1
28	0	0	0	0	1	3	0	0	0
29	0	0	1	0	1	3	0	0	0
30	0	0	2	0	1	3	0	0	0
31	0	0	0	1	1	3	-	0	-
32	0	0	1	1	1	3	-	0	0
33	0	0	2	1	1	3	0	0	1
34	0	1	2	1	1	3	-	0	0

takes about 1000 epochs to meet the error criterion in the above examples, the time for each epoch is small. Second, due to the distributed representation, only the receptive fields which respond to the training data are adjusted in each iteration. Third, the first-phase learning has built up the regularity in the training data such that the backpropagation in the second-phase learning just fine tunes the parameters based on the *priori knowledge* obtained in the first-phase learning.

Like the standard backpropagation algorithm, the procedure in our second-phase learning algorithm is a gradient descent procedure in which the parameters (i.e., the centers and widths of membership functions) are modified along the negative direction of the gradient of E [(25)] with respect to these parameters. Also it has been shown that Gaussian sum density estimator can approximate any probability density function to any desired degree of accuracy [36]. So it can be expected that the parameters will eventually converge to the values which minimize E to within some small fluctuations if there is adequate learning, adequate number of neurons and connections, and there is a deterministic rather than a stochastic relation between input and desired output. Since the proposed architecture has hidden units and nonlinear activation functions, the error surface may contain many local minima. So our hybrid learning algorithm is also bounded by all of the problems of any hill climbing procedure, such as the problem of local minima. This has not been shown to be a serious problem for standard backpropagation procedure in many practical applications or simulations [23]. The more general activation functions (i.e., bell-shaped functions) used in the proposed architecture have been used by other researchers [37], [38]. These empirical studies and the above simulations lead us to be optimistic about the local minimum problem. The poor local minima are rarely encountered if the network contains a few more neurons and connections than are required for a learning task [18]. Our simulation results also showed that increasing the number of fuzzy partitions has the effect of reducing the minimum error that can be achieved in some cases. Currently, the number of fuzzy partitions for each state variable is decided by the user. By adding extra conditions (e.g., the number of linguistic term nodes) other than the measure of model error on the training data into the performance criterion [39], the choice of the number of fuzzy partitions can be possibly automated.

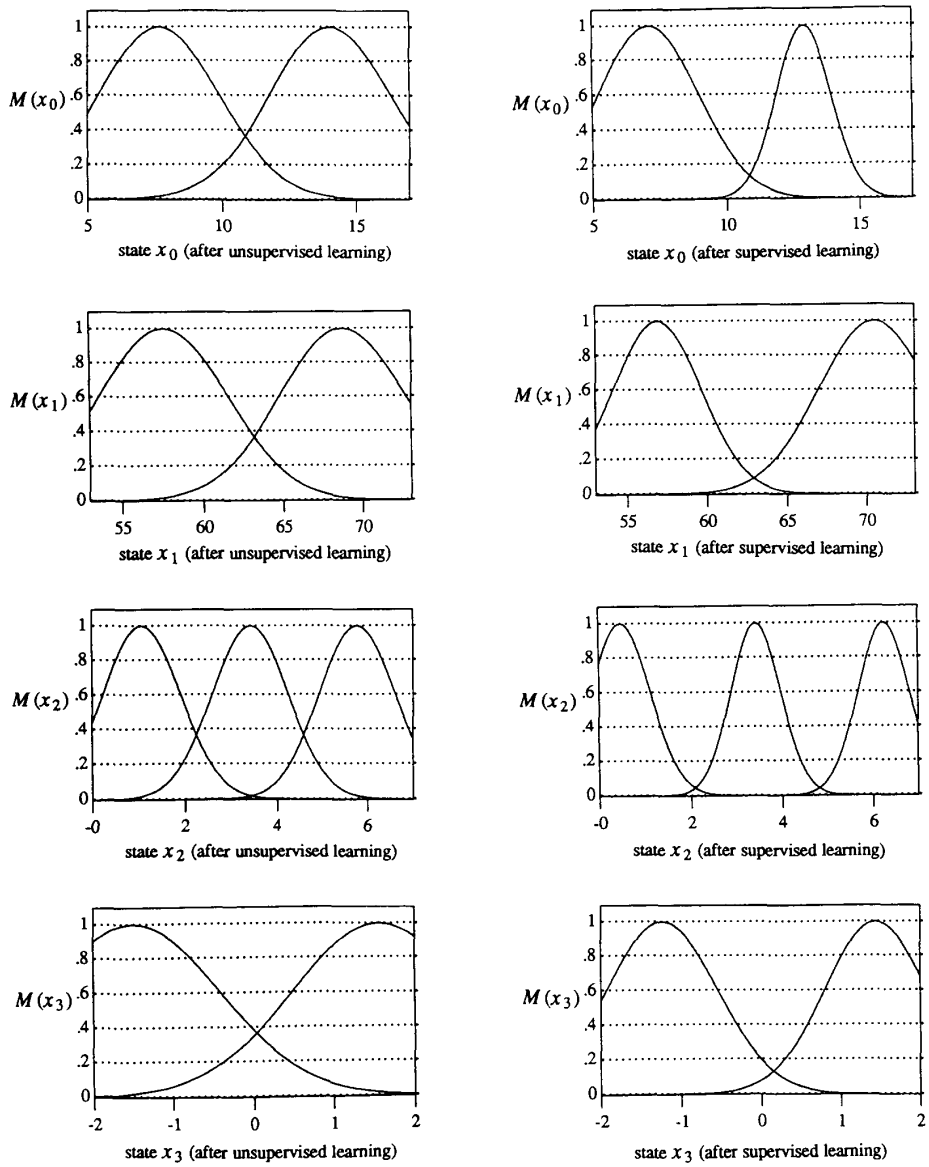


Fig. 11. Learned membership functions in Example 2.

VI. Conclusion

A general connectionist model of a fuzzy logic controller and decision-making system was proposed and presented. A hybrid learning scheme which combines a self-organized and supervised learning algorithms was developed for this model. The proposed model introduces the low-level learning power of neural networks into the fuzzy logic system and provides high-level human-understandable meaning to the normal connectionist architecture. Two examples were presented to demonstrate the two major applications of the proposed model: a fuzzy logic controller and a fuzzy logic decision-making system. Computer simulations were conducted on these two examples, and the results showed the superiority of the hybrid learning scheme over the traditional backpropagation learning on the normal multilayered feedforward neural networks, especially in the learning speed. Future research will focus on incorporating reinforcement learning techniques into our connectionist model to solve more complex problems in which supervised training data sets are expensive to obtain.

Appendix

The following proof technique follows the idea in [40]. For competitive learning,

$$\dot{w}_{ij}(t) = o_j^4(-w_{ij} + o_i^3). \tag{A.1}$$

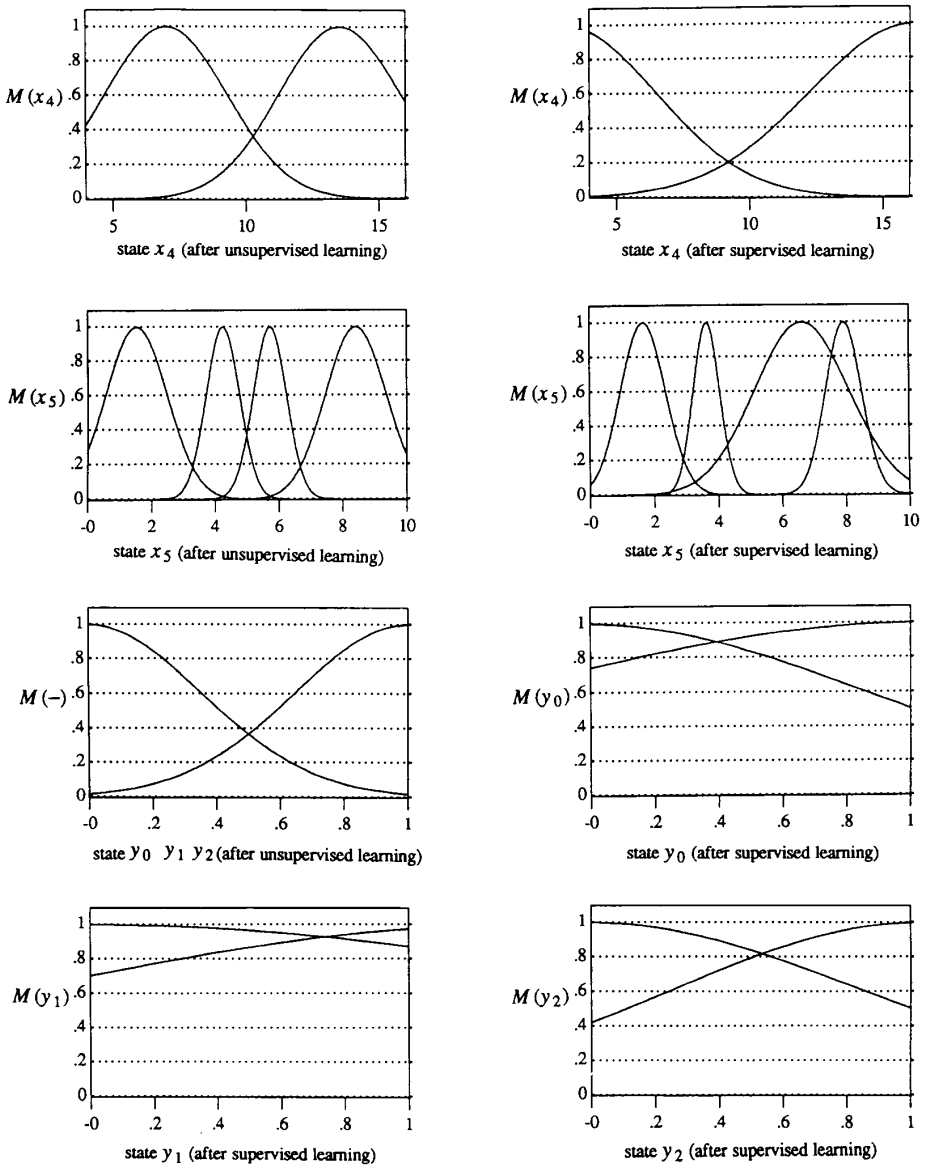


Fig. 11. *(Continued).*

This indicates the direction of modification of w_{ij}. To show the convergence of this rule, let $o_j^4 = \psi(\sum_i w_{ij} o_i^3)$ and let $\Phi(u)$ be the integral of ψ

$$\Phi(u) = \int_0^u \psi(x)dx. \tag{A.2}$$

Set the error function to be

$$E(w) = c[\frac{1}{2}\sum_i \sum_j \psi(\sum_i w_{ij} o_i^3) w_{ij}^2 - \frac{1}{2}\sum_i \sum_j \int w_{ij}^2 \psi'(\sum_i w_{ij} o_i^3) o_i^3 do_i^3 - \Phi(\sum_i w_{ij} o_i^3)], \tag{A.3}$$

then

$$\frac{\partial E}{\partial w_{ij}} = c\psi(\sum_i w_{ij} o_i^3) w_{ij} - \psi(\sum_i w_{ij} o_i^3) o_i^3 = c\psi(\sum_i w_{ij} o_i^3)(w_{ij} - o_i^3) = c o_j^4 (w_{ij} - o_i^3). \tag{A.4}$$

Hence,

$$\dot{w}_{ij} = -c\frac{\partial E}{\partial w_{ij}}. \tag{A.5}$$

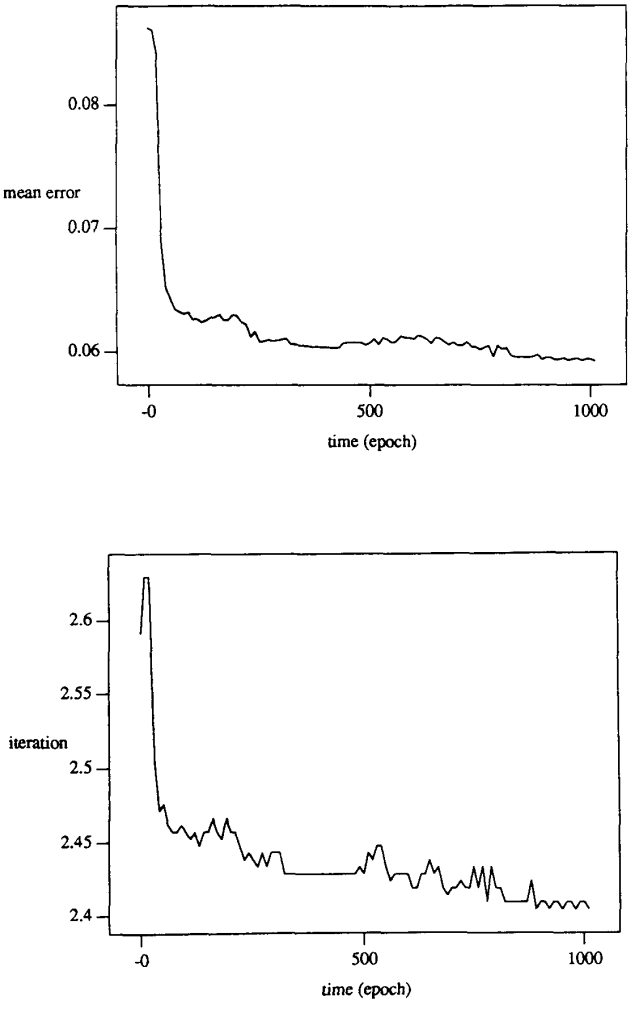


Fig. 12. Learning curves: mean error and mean iteration versus time (epoch) in Example 2.

This indicates that w_{ij} is modified along the negative direction of the gradient of E with respect to w_{ij}. Here it is expected that w_{ij}'s eventually converge to the values which minimize $E(w)$ to within some small fluctuations.

REFERENCES

[1] L. A. Zadeh, "Fuzzy sets," *Inform. Contr.*, vol. 8, pp. 338–353, 1965.
[2] M. Sugeno, Ed., *Industrial Applications of Fuzzy Control*. Amsterdam: North-Holland, 1985.
[3] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Part I & II," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-20, no. 2, pp. 404–435, 1990.
[4] L. A. Zadeh, "Fuzzy logic," *IEEE Comput. Mag.*, pp. 83–93, Apr. 1988.
[5] K. L. Self, "Fuzzy logic design," *IEEE Spectrum*, vol. 27, pp. 42–44 and 105, Nov. 1990.
[6] Y. F. Li and C. C. Lan, "Development of fuzzy algorithms for servo systems," *IEEE Contr. Syst. Mag.*, pp. 65–72, Apr. 1989.
[7] E. M. Scharf and N. J. Mandic, "The application of a fuzzy controller to the control of a multi-degree-freedom robot arm," in *Industrial Application of Fuzzy Control*, M. Sugeno, Ed. Amsterdam: North-Holland, 1985, pp. 41–62.
[8] S. Shao, "Fuzzy self-organizing controller and its application for dynamic processes," *Fuzzy Sets Syst.*, vol. 26, pp. 151–164, 1988.
[9] R. Tanscheit and E. M. Scharf, "Experiments with the use of a rule-based self-organizing controller for robotics applications," *Fuzzy Sets Syst.*, vol. 26, pp. 195–214, 1988.
[10] M. Sugeno and M. Nishida, "Fuzzy control of model car," *Fuzzy Sets Syst.*, vol. 16, pp. 103–113, 1985.
[11] T. Sejnowski and C. Rosenberg, "NETtalk: A parallel network that learns to read aloud," JHU/EECS-86/01, Tech. Rep., EECS Dep., Johns Hopkins Univ., 1986.
[12] K. Fukushima, S. Miyaka, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 826–834, 1983.
[13] S. Grossberg, "Cortical dynamics of three-dimensional form, color, and brightness perceptions," *Perception and Psychophys.*, vol. 41, no. 2, pp. 87–116, 1987.
[14] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 834–847, 1983.
[15] C. W. Anderson, "Learning to control an inverted pendulum using neural network," *IEEE Contr. Syst. Mag.*, pp. 31–36, Apr. 1989.
[16] F. C. Chen, "Back-propagation neural network for nonlinear self-tuning adaptive control," *Proc. IEEE Intelligent Machine*, pp. 274–279, 1989.
[17] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed rep-

resentations," in *Parallel Distributed Processing, Vol. 1.* Cambridge, MA: M.I.T. Press, 1986, pp. 77–109.
[18] G. E. Hinton, "Connectionist learning procedures," *Artif. Intell.*, vol. 40, no. 1, pp. 143–150, 1989.
[19] B. Kosko, "Unsupervised learning in noise," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 44–57, 1990.
[20] A. Amano and T. Aritsuka, "On the use of neural networks and fuzzy logic in speech recognition," in *Proc. 1989 Int. Joint Conf. Neural Networks*, 1989, pp. 301–305.
[21] C. C. Lee and H. R. Berenji, "An intelligent controller based on approximate reasoning and reinforcement learning," *Proc. IEEE Intelligent Machine*, pp. 200–205, 1989.
[22] B. Kosko, "Adaptive inference in fuzzy knowledge networks," in *Proc. 1987 Int. Joint Conf. Neural Networks*, 1987, pp. II 261–268.
[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing, Vol. 1.* Cambridge, MA: M.I.T. Press, 1986, pp. 318–362.
[24] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computat.*, vol. 1, pp. 281–294, 1989.
[25] W. Y. Huang and R. P. Lippmann, "Neural net and traditional classifiers," in *Neural Information Processing Systems.* New York: American Institute of Physics, 1988, pp. 387–396.
[26] M. Braae and D. A. Rutherford, "Fuzzy relations in a control setting," *Kybernetes,* vol. 7, no. 3, pp. 185–188, 1978.
[27] M. Togai and H. Watanabe, "Expert system on a chip: An engine for real-time approximate reasoning," *IEEE Expert Syst. Mag.*, vol. 1, pp. 55–62, 1986.
[28] T. Yamakawa and T. Miki, "The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process," *IEEE Trans. Comput.*, vol. C-35, no. 2, pp. 161–167, 1986.
[29] D. S. Touretzky and G. E. Hinton, "A distributed connectionist production system," CMU-CS-86-172, Tech. Rep., Dep. Comput. Sci., Carnegie Mellon Univ., 1986.
[30] S. I. Gallant, "Connectionist expert systems," *Commun. ACM,* vol. 31, no. 2, pp. 152–169, Feb. 1988.
[31] T. Kohonen, *Self-Organization and Associative Memory.* Berlin, Germany: Springer-Verlag, 1988, p. 132.
[32] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive Sci.,* vol. 9, pp. 75–112, 1985.
[33] S. Grossberg, "Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors," *Biol. Cybern.,* vol. 23, pp. 121–134, 1976.
[34] T. E. Vollmann, W. L. Berry, and D. C. Whybark, *Manufacturing Planning and Control Systems.* Homewood, IL: Richard D. Irwin, Inc., 1988, ch. 13.
[35] K. R. Baker, *Introduction to Sequencing and Scheduling.* New York: Wiley, 1974.
[36] H. W. Sorenson and D. L. Alspach, "Recursive Bayesian estimation using Gaussian sums," *Automatica*, vol. 7, no. 4, pp. 465–479, July 1971.
[37] S. Lee and R. M. Kil, "Multilayer feedforward potential function network," in *Proc. 1988 Int. Joint Conf. Neural Networks*, 1988, pp. I161-I171.
[38] J. A. Franklin, "Input space representation for refinement learning control," in *Proc. 1989 Int. Symp. Intelligent Contr.*, 1989, pp. 115–122.
[39] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal network construction with backpropagation," in *Advances in Neural Information Processing Systems 1.* Los Altos, CA: Morgan Kaufmann, 1989, pp. 177–185.
[40] S. Amari, "Neural theory of association and concept-formation," *Biol. Cybern.*, vol. 26, pp. 175–185, 1977.

Chin-Teng Lin (S'89) received the B.S. degree in control engineering from the National Chiao-Tung University, Taiwan, R.O.C., in 1986 and the M.S.E.E. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1989.

Currently, he is a candidate for the Ph.D. degree in the School of Electrical Engineering, Purdue University. His current research interests are neural networks, learning systems, fuzzy systems, intelligent control, artificial intelligence, and parallel processing.

Mr. Lin is a member of Tau Beta Pi and Eta Kappa Nu. He is also a student member of the IEEE Computer Society, the IEEE Robotics and Automation Society, and the IEEE Systems, Man, and Cybernetics Society.

C. S. George Lee (S'71–M'78–SM'86) received the B.S. and M.S. degrees in electrical engineering from Washington State University in 1973 and 1974, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 1978.

In 1978–1979, he taught at Purdue University, and in 1979–1985, at the University of Michigan. Since 1985, he has been with the School of Electrical Engineering, Purdue University, where he is currently a Professor of Electrical Engineering. His current research interests include computational robotics, intelligent multirobot assembly systems, and neural networks.

Dr. Lee was an IEEE Computer Society Distinguished Visitor in 1983–1986, the Organizer and Chairman of the 1988 NATO Advanced Research Workshop on Sensor-Based Robots: Algorithms and Architectures, and the Secretary of the IEEE Robotics and Automation Society in 1988–1990. He is Vice-President for Technical Affairs of the IEEE Robotics and Automation Society, a Technical Editor of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, an Associate Editor of the *International Journal of Robotics and Automation*, a co-author of *Robotics: Control, Sensing, Vision, and Intelligence* (New York: McGraw-Hill), and a co-editor of *Tutorial on Robotics*, second ed. (IEEE Computer Society Press). He is a member of Sigma Xi and Tau Beta Pi.