CHAPTER 13

NEURAL METHODS IN FUZZY SYSTEMS

13.1 INTRODUCING THE SYNERGISM

The complementary nature of fuzzy and neural systems forms a basis for their synergistic utilization. In the last chapter we examined ways of fuzzifying neurons and networks for the purpose of making them more expressive. In this chapter we focus on ways of endowing fuzzy systems with neuronal learning capabilities for the purpose of making them more adaptive. Although both fuzzy and neural approaches possess remarkable properties when employed individually, there are great advantages to using them synergistically.

Quite often, a strong point of fuzzy systems turns out to complement nicely a weakness—so to speak—of neural networks and conversely. This may not be so surprising, after all, since both neural and fuzzy approaches have had strong biological and cognitive inspirations. In the realm of human intelligence, there is certainly a synergism between the neuronal transduction and processing of sensory signals, on the one hand, and the cognitive, perceptual, and linguistic higher functions of the brain, on the other.

Consider, for example, the processes involved in our awareness of ambient temperature. The sensation of temperature is based on two kinds of *temperature receptors* in the skin—one signaling *warmth*, the other *cold* (Nicholls et al., 1992). Neural fibers from thousands of temperature receptors enter the spinal cord to form synapses with second-order neurons[1]. The second-order axons ascend into the *ventral* and *lateral spinothalamic* tract to end in the *medial* and *ventrobasal thalamus* (in the lower part of the brain) as shown in Figure 13.1. Third-order cells then carry the information to the *cerebral cortex*

[1] Higher-order biological neurons performing communication functions.

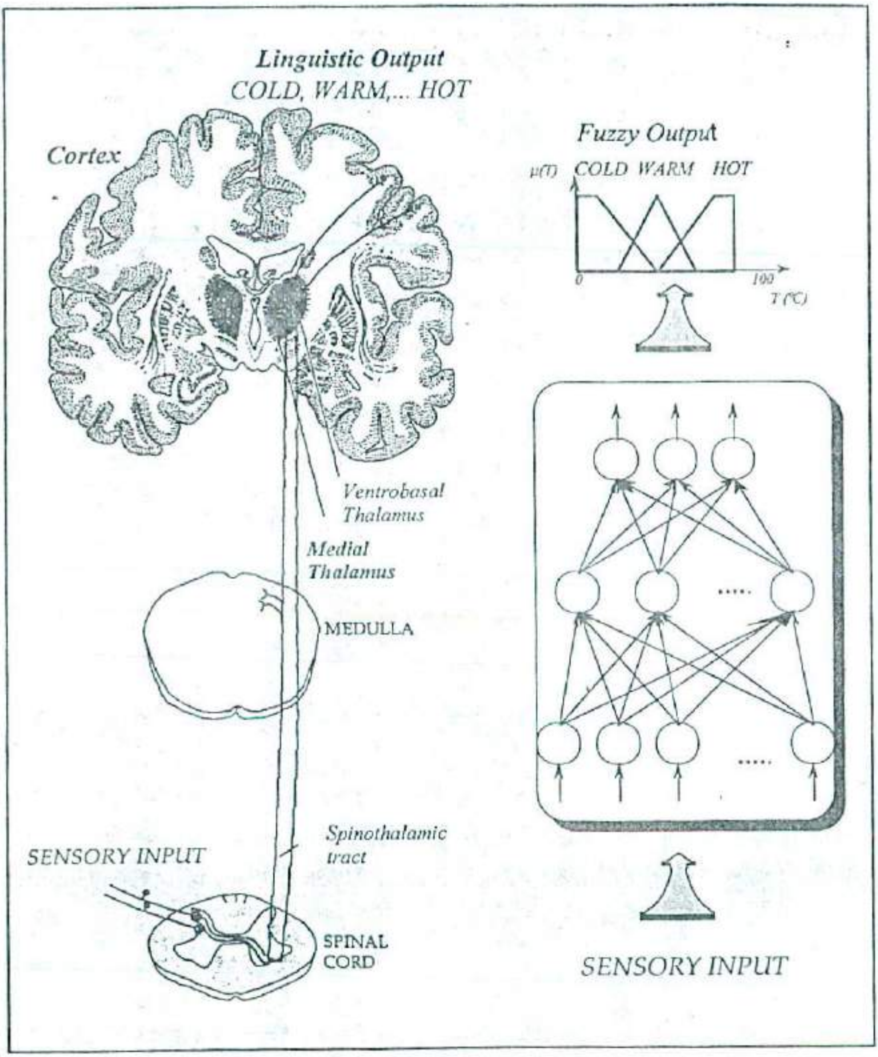


Figure 13.1 Sensory pathways take neuronal signals from temperature receptors in the skin through the spinal cord and the lower brain to the cerebral cortex, where they are ultimately transformed into linguistic categories. Fuzzy-neural hybrids are inspired by such biological-cognitive synergisms.

(the upper part of the brain). In the cerebral cortex, ultimately these temperature sensations get fused and expressed linguistically. Thus, a person may "know" that the temperature in the room is *COLD* or *WARM* or *HOT*, all linguistic (fuzzy) categorizations of the sensation of temperature.[2] This knowledge becomes a basis for human decisions and actions such as, for example, turning off the air-conditioning or a heating system. By analogy, we can train a neural network to cluster and map a set of temperature measurements from the ambience to a set of fuzzy values as shown in Figure 13.1. Of course, compared to the complexities and intricacies of the biological-cognitive system responsible for the sensation of temperature, our fuzzy-neural analog is at best very naive.

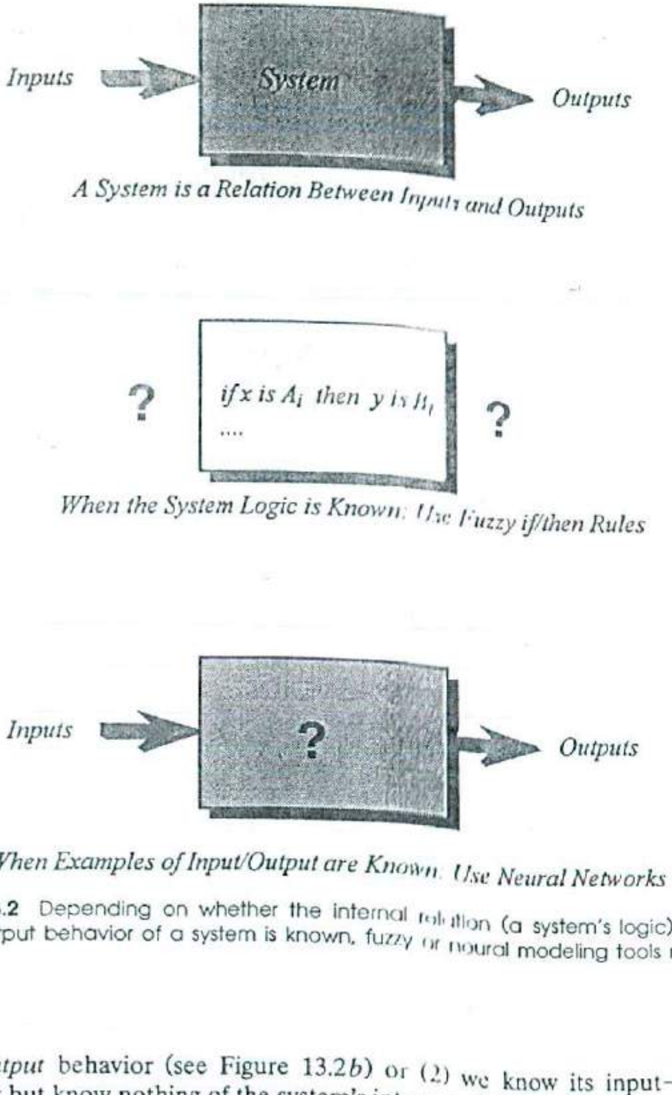
In this chapter we bring neural methods into fuzzy systems, both for the purpose of identifying (extracting) rules and membership functions and for adaptation of a fuzzy system (or linguistic description) to a changing physical system and its environment. The approach is known in the literature as *neural-network-driven fuzzy reasoning* (Takagi, 1992). For both *expert knowledge elicitation* and *adaptation*, the underlying strategy is, in essence, to *identify certain parameters of fuzzy systems and use neural networks to induce and/or adjust them*. Generally a fuzzy linguistic description of the kind we examined in Chapters 5 and 6 is computationally identical to a neural net, a fact theoretically proven by Buckley and Hayashi (1993), who demonstrated that neural nets can approximate continuous fuzzy controllers (and conversely) to any degree of accuracy.

Adaptation concerns the maintenance of a fuzzy linguistic description on the face of a changing process. The salient questions here have to do with how to adjust, over time, either the rules or what is involved within the rules, in order to better reflect changes in the actual physical system and its environment. Adaptation relates to the issue of *learning*. An adaptive system (that is, an *adaptive system description*) is one that can learn about the changes in the physical (target) system and modify its internals to improve the correspondence between the physical system and itself and/or its environment.

13.2 FUZZY-NEURAL HYBRIDS

In an abstract manner, a system can be viewed as shown in Figure 13.2*a*, a relation between inputs and outputs (where the relation is not necessarily a function, but a more general relation such as a *many-to-many* mapping). In Figure 13.2*b* and 13.2*c* we have two idealized extremes where either (1) we know exactly how the system should be working but have no example of its

[2] It is interesting to note that at a skin temperature of about 33°C (91.4°F) we are usually unaware of any temperature sensation. Raising or lowering skin temperature above this neutral point produces a sensation of *warming* or *cooling*.



A System is a Relation Between Inputs and Outputs

When the System Logic is Known: Use Fuzzy if/then Rules

When Examples of Input/Output are Known: Use Neural Networks

Figure 13.2 Depending on whether the internal relation (a system's logic) or the input–output behavior of a system is known, fuzzy or neural modeling tools may be chosen.

input–output behavior (see Figure 13.2*b*) or (2) we know its input–output behavior but know nothing of the system's internals (i.e., we have a black box) (see Figure 13.2*c*).

In the first case, it is convenient to write fuzzy *if/then* rules, at the appropriate level of precision, to describe (or prescribe) system behavior. In the second case, it is convenient to use the available input–output data to train artificial neural networks to model the internals of the system. Of course, in real-world systems we may have some examples of a system's input–output behavior and some knowledge of what is inside the black (or better yet "gray") box. Hence, we may utilize various hybrids of neural and fuzzy tools to successfully model the system. In the final analysis, however, our choice is made not by a commitment to a particular tool but a desire to adequately model the system at hand in a timely, reliable, and cost-effective manner.

The great array of system conditions that may be encountered calls for a variety of series and parallel combinations of fuzzy and neural systems. Consider the arrangement shown in Figure 13.3 which provides a means of inspecting and testing physically damaged components. Here a neural network is trained to receive three measurements as inputs (electrical and visual data from an automated test station testing electronic components for the purpose of eliminating physical defects (O'inca, 1994)). The input is mapped to two numerical values that serve as input to a fuzzy algorithm. The output of the neural module indicates the degree of a component's *physical damage* (a number between 0 and 1) and the *signal-to-noise ratio* (a number between 0 and 30). These two features are subsequently fed as inputs into a fuzzy system where fuzzy variables map signal-to-noise ratio and physical damage information to the quality of the component. The output of the fuzzy system is an action (decision) to accept or reject the component.

The benefits in using hybrid combinations of neural and fuzzy systems such as the one shown in Figure 13.3 are due to the fact that numerical measurements may actually provide too much detail to be effectively used

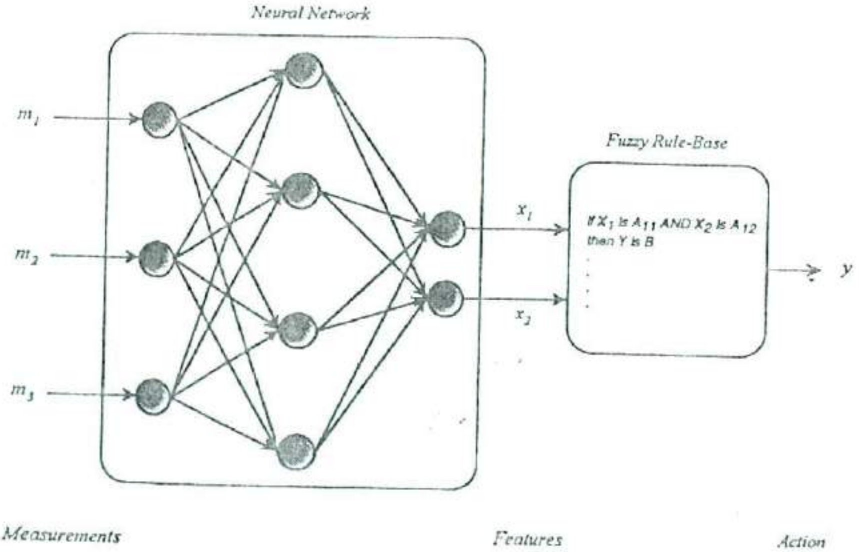


Figure 13.3 A hybrid system involving a neural network *in series* with a fuzzy system where measurements get mapped to features serving as inputs to the fuzzy system.

on-line (in addition to noise and other problems). Hence, neural filtering, smoothing, and mapping of numerical measurements to a feature space (e.g., *physical damage* and *signal-to-noise ratio*) may facilitate quick action by a fuzzy controller. Of course the arrangement can be reversed; that is, fuzzy processing can precede the neural network. Such an approach was taken by Yea and his coworkers in Japan in an interesting project involving odor discrimination where in order to discriminate amongst many kinds of odor species, a system has been developed using multiple gas sensors as sensory input to neural networks (Yea et al., 1994). When the system is presented with a number of inflammable gases, fragrant smells or even offensive odors, it is capable of an almost 100% discrimination of the different odors. The discrimination is performed in two steps: First, classification of the odor group performed by a fuzzy algorithm—that is, determining the groups of inflammable gases, fragrant smells, or offensive odors; second, discrimination of individual odor species in the classified group, performed by neural tools.

13.3 NEURAL NETWORKS FOR DETERMINING MEMBERSHIP FUNCTIONS

An interesting and often advantageous feature of fuzzy systems is that they allow for rather flexible categorization of a domain of interest. For example, when a problem calls for a small number of categories of temperature, we define *SMALL*, *MEDIUM*, and *LARGE* as the values of the fuzzy variable *temperature*, instead of say 100 categories of natural numbers taking us from 1°C to 100°C. For each and every linguistic value a unique membership function analytically describing the degree of membership to the fuzzy value of each individual crisp element of the universe of discourse is sought. The problem of determining membership functions has occupied a central importance in the history of fuzzy logic with a number of subjectivist, statistical, and (more recently) neural approaches being proposed.

Membership function determination may be viewed as a data clustering and classification problem. Hence, neural clustering and classification algorithms can be brought to bear to solve this problem as illustrated in Figure 13.4. When multidimensional data are clustered, we can extract either one-dimensional membership functions based on a distance metric δ (as shown in the figure) or obtain multidimensional membership functions modeling fuzzy relations (i.e., *if/then* rules). A typical use of neural networks for producing membership functions involved a two-stage process: *clustering* and *fuzzification* (Adeli and Hung, 1995). The first stage is essentially a *classification* stage where a neural network is used to classify or cluster domain data into a certain number of clusters. The second stage is a fuzzification process where fuzzy membership values are assigned (to each training instance) in the set of classified clusters [see also Travis, 1994)]. Of course the problem of membership function determination is not totally separate from the problem

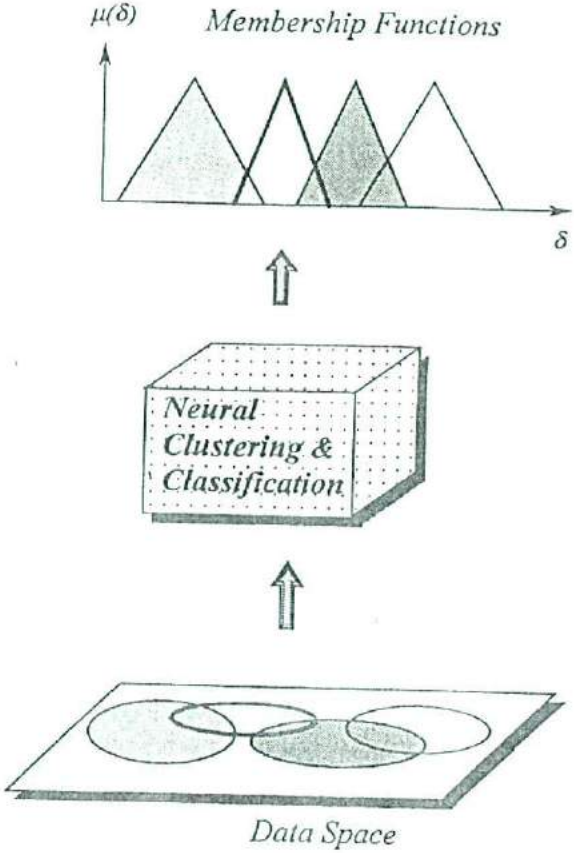


Figure 13.4 Clustering approaches are used to determine membership functions in data-rich applications.

of identifying rules (only the alter one is much more difficult). In practice, both the determination of membership functions and the extraction of rules proceed through some kind of clustering.

The methods used in the categorization of a universe of discourse are typically based on some type of Kohonen or unsupervised learning network. A pattern clustering method based on the Kohonen feature mapping algorithm and the backpropagation multilayer perceptron has been used for membership function determination by Pham and Bayro-Corrochano (1994). The method is applied first to the training data set to divide it into labeled clusters using the Kohonen algorithm and a cluster labeling procedure. The data clusters are then employed to train a three-layer perceptron. The approach is self-organizing by virtue of the Kohonen algorithm, and it produces fuzzy outputs as a consequence of the backpropagation network.

Kuo, Cohen, and Kumara at Penn State have taken a similar yet different approach in developing a novel self-organizing and self-adjusting fuzzy modeling approach with learning capabilities (Kuo et al., 1994). Basically, their approach consists of two stages: a *self-organizing* and a *self-adjusting* stage. In the first stage, the input data are divided into several groups by applying Kohonen's feature maps. Gaussian distribution functions are employed as the standard form of the membership functions. Statistical tools are used to determine the center and width of the membership function for each group. Error backpropagation (see Section 13.5) fine-tunes the parameters involved. Feedforward neural estimation for membership function determination and fuzzy classification have also been investigated by Purushotaman and Karayiannis at the University of Houston (Purushothaman and Karayiannis, 1994), while Higgins and Goodman at MIT developed a different method for learning membership functions and rules from a set of examples (Higgins and Goodman, 1994). Their method is a general function approximation system using a three-step approach: first, learning the membership functions and creating a cell-based rule representation; second, simplifying the cell-based rules using an information-theoretic approach for induction of rules from discrete-valued data; and, finally, constructing a neural network to compute the function value given its independent variables.

A typical use of neural networks for producing membership functions involved a two-stage process. The first stage is essentially a *classification* stage where a neural network is used to classify or cluster domain data into a certain number of clusters. The second stage is a *fuzzification process* where the fuzzy membership values for each training instance in the set of supports, classified clusters, are evaluated. Let us look at the *Adeli–Hung algorithm* (AHA) for determining membership functions (Adeli and Hung, 1994).

Determining Membership Functions Through the Adeli–Hung Algorithm

Suppose that our data consist of N training instances $X_1, X_2, \ldots, X_N$ and we have M patterns in each training instance, $X_i = [x_{i_1}, x_{i_2}, \ldots, x_{i_M}]$. The *mean vector* of these instances may be defined as

$$\overline{X}_N = \frac{1}{N} \sum_{i-1}^{N} X_i \qquad (13.3\text{-}1)$$

For $N + 1$ training instances the mean vector is found from the mean $\overline{X}_N$

and the instance X_{N+1} as follows:

$$\begin{aligned}\overline{X}_{N+1} &= \frac{1}{N+1}\sum_{i=1}^{N+1} X_i \\ &= \frac{1}{N+1}\left[\sum_{i=1}^{N} X_i + X_{N+1}\right] \\ &= \frac{1}{N+1}\left[N\overline{X}_N + X_{N+1}\right] \\ &= \frac{N}{N+1}\overline{X}_N + \frac{1}{N+1}X_{N+1} \qquad (13.3\text{-}2)\end{aligned}$$

Classification by AHA is performed using a topology-and-weight-change, two-layer (flat) neural network where the number of input nodes equals the number of patterns (M) in each training instance and the number of output nodes equals the number of clusters.

The algorithm uses a neural network $NN(M, 1)$ with M inputs and an as yet undetermined number of outputs. The first training instance gets assigned to the first cluster. If the second instance is classified to the first cluster, the output node representing the first cluster becomes active. If the second training instance is classified as a new cluster, an additional output node is added to the network, and so on, until all training instances are classified.

To perform the classification in AHA, a function $diff(X, C)$ is defined, called the *degree of difference*, representing the difference between a training instance X and a cluster C in a $NN(M, P)$ network (P indicates the number of output nodes or equivalently the number of classes). This function maps two given vectors (X and C) to a real number (*diff*). The patterns of each cluster (means of the patterns of the instance in the cluster) are stored in the weights of the network during the classification process. The following procedure for classifying a training instance into an active or new cluster is used in AHA:

Step 1. Calculate the degree of difference, $diff(X, C_i)$, between the training instance, X, and each cluster, C_i. A Euclidean distance is used (in image recognition applications) and the function $diff(X, C_i)$ becomes

$$diff(X, C_i) = \sqrt{\sum_{j=1}^{M} (x_j - c_{ij})^2} \qquad (13.3\text{-}3)$$

Step 2. Find the smallest degree of difference, $diff_{\min}(X, C_i)$, and make the cluster with the smallest degree of difference an active cluster:

$$C_{\text{active}} = \{C|\min\{diff(X, C_i)\},\ i = 1, 2, \ldots, P\} \qquad (13.3\text{-}4)$$

Step 3. Compare the value of $diff_{\min}$ with a predefined a threshold value κ. If the value of $diff_{\min}$ is greater than the predefined threshold, the training instance is classified as a new cluster (at this point one more output node is turned on).

$$C_{\text{new}} = X \qquad \text{if } k < \min\{diff(X, C_i),\ i = 1, 2, \ldots, P\} \quad (13.3\text{-}5)$$

Suppose the given N training instances have been classified into P clusters. Let us use the symbols C_j to denote the jth cluster and use U to denote the set of all clusters. If the clusters are completely disjoint, each instance in the training set belongs to only one of the classified clusters and a binary matrix Z can be used to record the cluster of each instance. If the instance i belongs to the j cluster we have $z_{ij} = 1$, while if it does not belong we have $z_{ij} = 0$. On the other hand, if the classified clusters are partly overlapping, a given instance in the training set may belong to more than one cluster. Hence the boundaries of the classified clusters are fuzzy rather than crisp. The same binary matrix Z may be used to record the cluster of each instance. The prototype for each cluster is defined as the mean of all instances in that cluster, and the degree of membership of each instance in the cluster is based on how similar this instance is to the prototype one. The similarity can be defined as a function of distance between the instance and the prototype of the cluster. If there are n_p instances in a cluster p, the pattern vector of the ith instance in the cluster p is $X_i^p = [x_{i_1}^p, x_{i_2}^p, \ldots, x_{i_M}^p]$. Then, the vector of the prototype instance (the mean of all instances) in cluster p is defined as

$$C_p = [c_{p_1}, c_{p_2}, \ldots, c_{p_M}] = \frac{1}{n_p} \sum_{i-1}^{n_p} X_i^p \qquad (13.3\text{-}6)$$

where $c_{p_j} = (1/n_p)\Sigma_{i=1}^{n_p} x_{ij}^p$ and $j = 1, 2, \ldots, M$. Using triangular-shaped membership functions (over the *diff* universe of discourse) the fuzzy membership value of the ith instance in the p cluster is defined as

$$\mu_p(X_i^p) = f\left[D^w(X_i^p, C_p)\right]$$
$$= \begin{cases} 0 & \text{if } D^w(X_i^p, C_p) > \kappa \\ 1 - \dfrac{D^w(X_i^p, C_p)}{\kappa} & \text{if } D^w(X_i^p, C_p) \le \kappa \end{cases} \qquad (13.3\text{-}7)$$

where a predefined threshold value κ is used as crossover value. The similarity function is defined as the weighted norm $D^w(X_i^p, C_p)$. The weighted norm in the Adeli–Hung algorithm is defined as the Euclidean distance:

$$D^w(X_i^p, C_p) = \| w_p(X_i^p, C_p) \|^w = \sqrt{\sum_{j=1}^{M} (x_{ij}^p - c_{pj})^2} \quad (13.3\text{-}8)$$

In image recognition problems, a value of 1 is used for the weight parameters w and w_p. If the Euclidean distance for a given instance is less than the crossover value κ, the instance belongs to the cluster p to a degree given by the membership value.

13.4 NEURAL-NETWORK-DRIVEN FUZZY REASONING

In fuzzy systems employing more than three or four fuzzy variables, it may be practically difficult to formulate fuzzy *if/then* rules, and it would be desirable if they could be extracted automatically out of data from the physical system being modeled. The problem of inducing (extracting) fuzzy rules has been addressed by several researchers and is still undergoing intense investigation (Kosko, 1992; Takagi, 1991; Hayashi et al., 1992; (Keller and Tahani, 1992; (Keller et al., 1994; Khan, 1993; Li and Wu, 1994; Wang, 1994; Wang and Mendel, 1992; Nie, 1994; Jang and Sun, 1995; Werbos, 1992; Yager, 1994; Blanco et al., 1995). In an important paper published in 1991, Matsushita Electric engineers Tagaki and Hayashi presented a comprehensive approach for the induction and tuning of fuzzy rules, known as *neural-network-driven fuzzy reasoning* or the *Takagi–Hayashi (T–H) method*.

Consider the situation shown in Figure 13.5 where we have the data space of two inputs, x_1 and x_2 (e.g., two measurements obtained from sensors), knowledge of the target or desirable output, and a nonlinear partition of this space in three regions. These regions correspond to three fuzzy *if/then* rules. The identified rules R_1, R_2, and R_3 are of the Sugeno variety (see Chapter 6); that is, their consequent is a functional mapping of the antecedent variables, with the mapping actually being performed by specially trained neural networks.[3] The Takagi–Hayashi method consists of three major parts:

Part 1: Partitions the control or decision hypersurface into a number of rules.

Part 2: Identifies a given rule's LHS (antecedent) values (i.e., determines their membership functions).

[3] A potential drawback of the T–H method as well as most similar methods is that one has to decide in advance the possible number of rules—for example, three rules in this case.

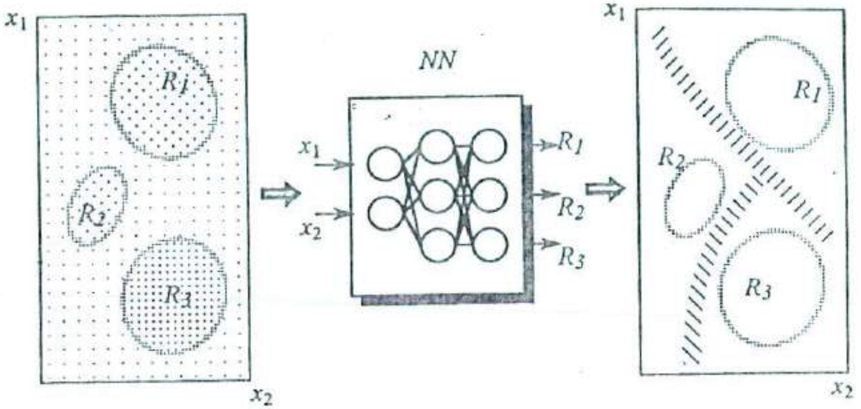


Figure 13.5 Schematic of the Hayashi–Takagi method for extracting fuzzy rules.

Part 3: Identifies a given rule's RHS (consequent) values (the amount of control for each control rule).

Part 1 determines the number of fuzzy inference rules through clustering performed on the data available. Part 2 employs a neural network to derive the membership function for each rule (it therefore identifies the LHS of rules). The T–H method combines all the variables (x_1 and x_2, for example) in the LHS and is based on the theoretical result that an arbitrary continuous function is equivalent to a neural network having at least one hidden layer. Buckley and Hayashi (1993) have shown the computational equivalence between continuous functions, regular neural nets, fuzzy controllers, and discrete fuzzy expert systems and have shown how to build hybrid neural nets numerically identical to a fuzzy controller or a discrete fuzzy expert system. Part 3 of the T–H method determines the RHS parts using neural networks with supervised learning (supervised by the learning data and the control value for each rule as in *Part 2*).

Sugeno-type rules are used (see Chapter 6) where the output is a function of the inputs. Sugeno rules are typically of the form

$$\text{if } x_1 \text{ is } A_1 \; AND \, x_2 \text{ is } A_2 \ldots, \text{ then } y = f(x_1, \ldots, x_n) \qquad (13.4\text{-}1)$$

where f is a function of the inputs $x_1, \ldots, x_n$. In the T–H method this function has been replaced by a neural network. For example, an induced rule would be of the form

$$\text{if } (x_1, x_2) \text{ is } A^s, \text{ then } y^s = NN_s(x_1, x_2) \qquad (13.4\text{-}2)$$

where $\mathbf{x} = (x_1, x_2)$ is the vector of inputs and $y^s = NN_s(x_1, x_2)$ is a neural

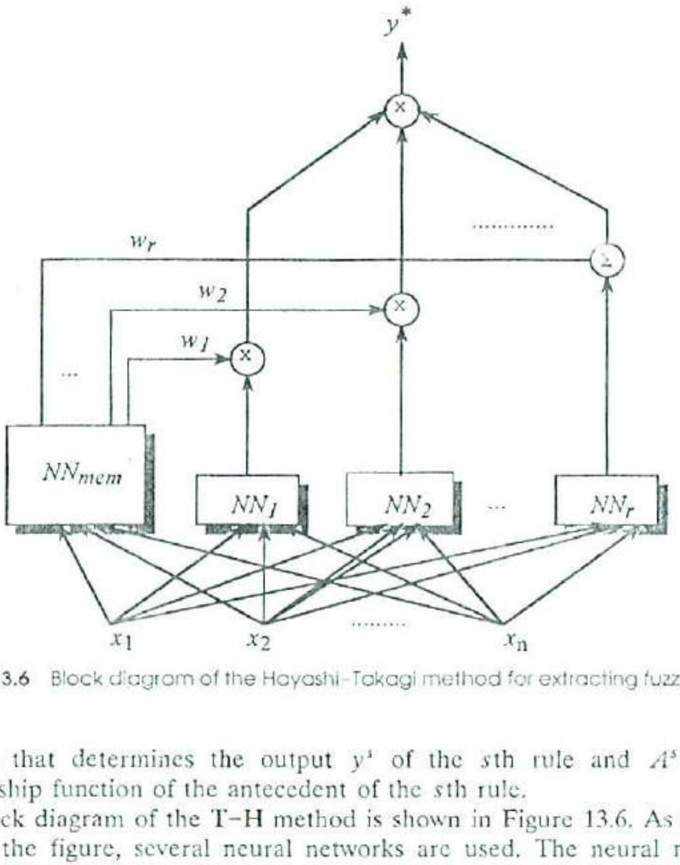


Figure 13.6 Block diagram of the Hayashi–Takagi method for extracting fuzzy rules.

network that determines the output y^s of the sth rule and A^s is the membership function of the antecedent of the sth rule.

A block diagram of the T–H method is shown in Figure 13.6. As may be seen in the figure, several neural networks are used. The neural network labeled NN_{mem} is responsible for generating the membership functions of the antecedents of rules while networks $NN_1, NN_2, \ldots, NN_r$ determine the consequent parts. Networks $NN_1, NN_2, \ldots, NN_r$ provide the RHS function of Sugeno rules shown in equation (13.4-1). In actual applications these are three-layer networks trained by backpropagation. As seen in Figure 13.6, the overall system weighs the output of the RHS networks by the membership values of LHS and computes a final output value. The following eight steps constitute the outline of the procedure used in the Takagi–Hayashi method:

Step 1. We define y_i as the output and define x_j, $j = 1, 2, \ldots, k$, as the input variables. Inputs x_j, $j = 1, 2, \ldots, m$, $m \leq k$, that are related to the observed value of the output are selected by a neural network through a backward elimination method using sum of squared errors as a cost function for the purpose of eliminating input variables attributed to noise. It is important to select only those input variables that have significant correlation to the observed values.

Step 2. The input–output population n is divided into *training data* (TRD) of n_t) and *checking data* (CHD of n_c), where $n = n_t + n_c$.

Step 3. The TRD is partitioned into r groups using a clustering method. Each partition is labeled as R^s, $s = 1, 2, \ldots, r$, and the data within the ith partition is expressed as $(\mathbf{x}_i^s, y_i^s)$, where $i = 1, 2, \ldots, (n_t)^s$ and $(n_t)^s$ are the TRD numbers in each R^s. Partitioning the data space into r partitions implies that the number of inference rules will be taken to be r.

Step 4. The antecedent part of each rule is identified through NN_{mem} (the neural network generating the membership functions, see Figure 13.6). If $\mathbf{x}_i$ are the values for the input layer of NN_{mem}, the weights w_i^s are assigned as the supervised data for the output layer, where

$$w_i^s = \begin{cases} 1, & \mathbf{x}_i \in R^s \\ 0, & \mathbf{x}_i \notin R^s \end{cases} \quad i = 1, \ldots, n_t; i = 1, \ldots, r \qquad (13.4\text{-}3)$$

The network NN_{mem} is trained to infer weights w_i^s given an input vector $\mathbf{x}$. NN_{mem} thus becomes capable of computing the degree of membership $\hat{w}_i^s$ of each training data item $\mathbf{x}_i$ to the rule (or partition) R^s. The membership function of the antecedent A^s of the sth rule is defined as the inferred value $\hat{w}_i^s$—that is, the output of NN_{mem}:

$$\mu_{A^s}(\mathbf{x}_i) \equiv \hat{w}_i^s, \qquad i = 1, 2, \ldots, n \qquad (13.4\text{-}4)$$

Step 5. After identifying the antecedent membership function in step 4, we now identify the *consequent* part of the Sugeno fuzzy *if/then* rules we are looking for. The RHS of each rule is expressed by the input–output relationship. Inputs $x_{i1}^s, \ldots, x_{im}^s$ and outputs y^s, $i = 1, 2, \ldots, (n_t)^s$, from the training data are used as input–output pairs for training the NN_s neural network that models the consequent of the sth rule. Subsequently the checking data $x_{i1}, \ldots, x_{im}$, $i = 1, 2, \ldots, n_c$, are used as input and the sum of squared errors is formed:

$$\Theta_m^s = \sum_{i=1}^{n_c} \left[y_i - u_s(\mathbf{x}_i) \cdot \mu_{A^s}(\mathbf{x}_i) \right]^2 \qquad (13.4\text{-}5)$$

where $u_s(\mathbf{x}_i)$ is the calculated output of NN_s, y_i is the target output for the network, and Θ_m^s is sum of squared errors. The sum can also be computed after weighing by $\mu_{A^s}(\mathbf{x}_i)$; that is,

$$\Theta_m^s = \sum_{i=1}^{n_c} \mu_{A^s}(\mathbf{x}_i) \cdot \left[y_i - u_s(\mathbf{x}_i) \cdot \mu_{A^s}(\mathbf{x}_i) \right]^2 \qquad (13.4\text{-}6)$$

Takagi and Hayashi use an index to decide the best iteration number during training (to prevent "overlearning" or memorization):

$$I^s = \frac{n_c}{(n_t)^s + n_c} \sum_{i=1}^{(n_t)^s} [y_i - u_s(\mathbf{x}_i)]^2 + \frac{(n_t)^s}{(n_t)^s + n_c} \sum_{j=1}^{n_c} [y_j - u_s(\mathbf{x}_j) \cdot \mu_{A^s}(\mathbf{x}_j)]^2 \qquad (13.4\text{-}7)$$

If the sth network has overlearned, the error of the TRD becomes small but the error of the CHD becomes large, suggesting that the optimum number of iterations is the one that gives the smallest I^s in equation (13.4-6).

Step 6. In this step a number of variables may be eliminated from the consequent through a backward elimination method. Out of the m input variables of a network inferring the consequent of a rule, one (e.g., x^p) is arbitrarily eliminated, and the neural network for each consequent is trained using the TRD as in step 5. Equation (13.4-8) below gives the squared error Θ_{m-1}^{sp} of the control value of the sth rule in the case where x^p has been eliminated. This sum squared error can be estimated using the *checking data*:

$$\Theta_{m-1}^{sp} = \sum_{i=1}^{n_c} [y_i - u_s(\mathbf{x}_i) \cdot \mu_{A^s}(\mathbf{x}_i)]^2, \qquad p = 1, 2, \ldots, m \quad (13.4\text{-}8)$$

After comparing equations (13.4-6) and (13.4-8) and $\Theta_m^s > \Theta_{m-1}^{sp}$, the significance of the eliminated variable x^p can be considered minimal, and x^p can be discarded.

Step 7. The operations in step 6 are carried out for the remaining $m - 1$ input variables until it is no longer true that $\Theta_m^s > \Theta_{m-1}^{sp}$ for any of the remaining input variables. The model that gives the minimum Θ^s value is the best-trained neural network for the sth rule.

Step 8. Equation (13.4-8) below gives the final control value y_i^*:

$$y_i^* = \frac{\Sigma_{s=1}^{r} \mu_{A^s}(\mathbf{x}_i) \cdot \{u_s(\mathbf{x}_i)\}_{\text{inf}}}{\Sigma_{s=1}^{r} \mu_{A^s}(\mathbf{x}_i)}, \qquad i = 1, 2, \ldots, n \quad (13.4\text{-}8)$$

where $[u_s(\mathbf{x}_i))_{\text{inf}}$ is an inferred value obtained when CHD is substituted in the best *NN* obtained in step 7.

It should be noted that the T–H method allows for nonlinear partitioning of the input space and hence the identification of nonlinear membership functions. Each cluster of input data corresponds to an *if/then* Sugeno-type rule as shown in (13.4-2). Although these rules individually make a good fit for data similar to what they have been trained on, a gradual fitting to multiple rules has to be performed for data near the boundary region (see Figure 13.5). The following example [taken from Hayashi et al. (1992)] serves to illustrate the T–H method.

Example 13.1 COD Density Estimation. Data of *chemical oxygen demand* (COD) density in Japan's Osaka Bay taken over a 10-month interval were used by Takagi and Hayashi (1992) to test their neural-network-driven fuzzy reasoning method. In this application of the T–H method the input–output variables are

y	COD density (ppm)
x_1	Water temperature (°C)
x_2	Transparency (m)
x_3	Dissolved oxygen density (ppm)
x_4	Salinity (%)
x_5	Filtered COD density (ppm)

In accordance with step 2 in the T–H method, the data were divided in training and checking data as shown in Figure 13.7. Thirty-two data points were used for estimation, while 12 data points were used for testing. Performing a backward elimination experiment suggested the use of all input variables for estimation.

For determining the membership functions of the antecedent sets (i.e., NN_{mem}), a four-layer network with five input nodes in the input layer, two

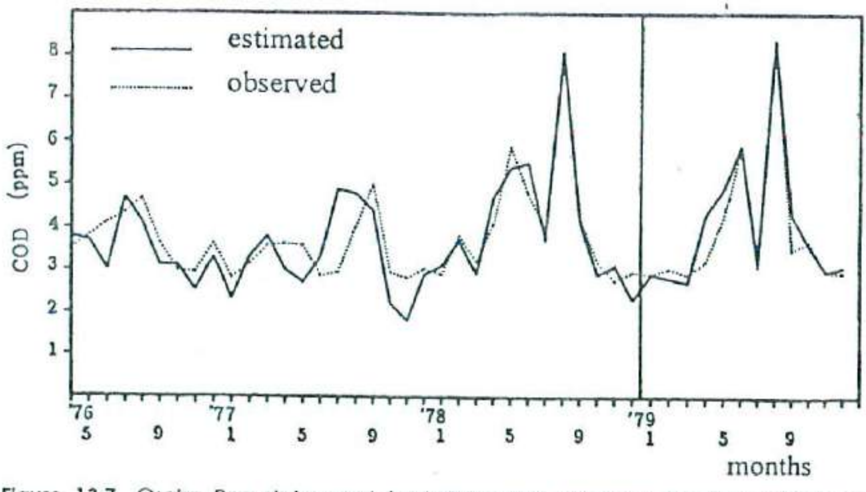


Figure 13.7 Osaka Bay data used for training and checking the Hayashi\Takagi method. (Takaki and Hayashi, 1991).

hidden layers with 12 nodes each, and two output nodes in the output layer was used. Similarly, for determining the consequent part of rules, four-layer networks were used with m input nodes in the input layer ($m = 5, 4, \ldots$), two hidden layers with 12 nodes each, and an output layer with only one node (see Figure 13.6).

Network training took 1500–2000 iterations, and the following rule structure was finally identified:

$$R_1: \quad \text{if } (x_1, x_2, x_3, x_4, x_5) \text{ is } A^1, \text{ then } y^1 = NN_1(x_1, x_2, x_3, x_4, x_5)$$

$$R_2: \quad \text{if } (x_1, x_2, x_3, x_4, x_5) \text{ is } A^2, \text{ then } y^2 = NN_2(x_1, x_2, x_3, x_5)$$

The estimated COD density by the above system was in very good agreement with observed data, and it performed better in comparison with results obtained by other methods. □

13.5 LEARNING AND ADAPTATION IN FUZZY SYSTEMS VIA NEURAL METHODS

In recent years Nomura, Hayashi, and Wakami proposed an approach to fuzzy system adaptation utilizing the gradient-descent error minimization we saw in connection with backpropagation in Chapter 8. A parameterized description of a fuzzy system with symmetric, triangular-shaped membership functions for inputs and crisp outputs was developed, and error minimization through gradient-descent was used (Nomura et al., 1994; Wang, 1994; Jang and Sun, 1995). Ichihashi et al. (1993) used gradient descent with exponential membership functions. Guély and Siarry (1993) have solved the problem more generally—that is, for symmetric as well as nonsymmetric antecedent membership functions and different connectives and consequent forms.

Most fuzzy system adaptation approaches rely on gradient-descent optimization. As is the case in neural learning, an objective function E is sought to be minimized:

$$E = \tfrac{1}{2}(y^r - y)^2 \qquad (13.5\text{-}1)$$

where y is the output of the fuzzy system and y^r is the reference (target) system output.

Consider the ith zero-order Sugeno rule of a system having n such rules ($i = 1, \ldots, n$):

$$R_i: \quad \text{if } x_1 \text{ is } A_{i1} \ AND \ \cdots \ AND\, x_m \text{ is } A_{im} \text{ then } y \text{ is } w_i \quad (13.5\text{-}2)$$

where $A_{i1}, \ldots, A_{im}$ are the fuzzy values of the LHS of the rule and w_i is a constant in the consequent of the ith rule. In a fuzzy system, we combine fuzzy *if/then* rules like the one above to perform a mapping from fuzzy sets of the LHS universe of discourse to constants in the RHS. Let μ_i be the membership function for the fuzzy relation of the ith rule. Then the output of a simplified fuzzy reasoning approach, y, can be obtained through equations

$$\mu_i = \prod_{j=1}^{m} A_{ij}(x_j) \tag{13.5-3}$$

and

$$y = \frac{\sum_{i=1}^{n} \mu_i \cdot w_i}{\sum_{i=1}^{n} \mu_i} \tag{13.5-4}$$

Using input–output data and a gradient-descent algorithm, we can optimize the w_i's by minimizing an objective function E such as the one given by equation (13.5-1). Let us rewrite this squared error function as

$$E = \tfrac{1}{2}(y^{rp} - y^p)^2 \tag{13.5-5}$$

where y^{rp} is the target for the pth input data $(x_1^p, \ldots, x_m^p)$ and y^p is the calculated output of the system corresponding to the same input data. The learning rule for the real numbers in the RHS of rules (13.5-2) is

$$w_i(t' + 1) = w_i(t') - K \cdot \frac{\partial E}{\partial w_i} \tag{13.5-6}$$

where t' is the number of iteration of learning. Following Nomura et al. (1994) and using the above error function (13.5-5) in (13.5-6), we express the w_i update as

$$w_i(t' + 1) = w_i(t') - K \cdot \frac{\mu_i^p}{\sum_{i=1}^{n} \mu_i^p}(y^p - y^{rp}) \tag{13.5-7}$$

where μ_i^p is the membership value of the ith rule corresponding to the pth input–output example and K is a constant.

Using input–output examples with learning rule (13.5-7) repeatedly, the RHS numbers w_i are updated so as to minimize the error function, ultimately reaching a global minimum since $\partial^2 E/\partial w_i^2 \geq 0$ is obtained for all rules.

As with *neural-network-driven fuzzy reasoning* in the previous section, in this adaptation approach too one has to come up *a priori* with the optimal number of rules, often through a trial-and-error approach. A number of researchers are proposing various genetic approaches to address this issue [see Nomura et al. (1994), Pedrycz (1995), Perneel (1995)][4].

[4] See Chapter 17 of this book.

Using gradient descent, the membership functions of the LHS of rules (15.4-2) may be tuned to better reflect the problem at hand. Consider the symmetric triangular membership function for the jth antecedent of the ith rule shown in Figure 13.8. Such a membership function can be represented by the peak a_{ij} and the support b_{ij}, and therefore the entire rule (13.5-2) can be parameterized through the peaks of antecedent values a_{ij}, their support b_{ij}, and w_i.

Following Guély and Siarry (1993), let us address adaptation for rules with symmetric triangular membership functions in the LHS, product interpretation of *AND*, center of area output calculation, and constant outputs as in rule (13.5-2). As seen in Figure 13.8*a*, the symmetric triangular membership functions in the LHS are given by

$$\mu_{ij}(x_j) = \begin{cases} 1 - \dfrac{|x_j - a_{ij}|}{5b_{ij}}, & \text{if } |x_j - a_{ij}| \leq \dfrac{b_{ij}}{2} \\ 0, & \text{otherwise} \end{cases} \qquad (13.5\text{-}8)$$

As before, we want to adapt the parameters (a_{ij}, b_{ij}, w_i). Let p denote the number of training samples, and y'^p the training sample output. Using gradient descent means that our peak parameters, for example, will be updated in the following manner:

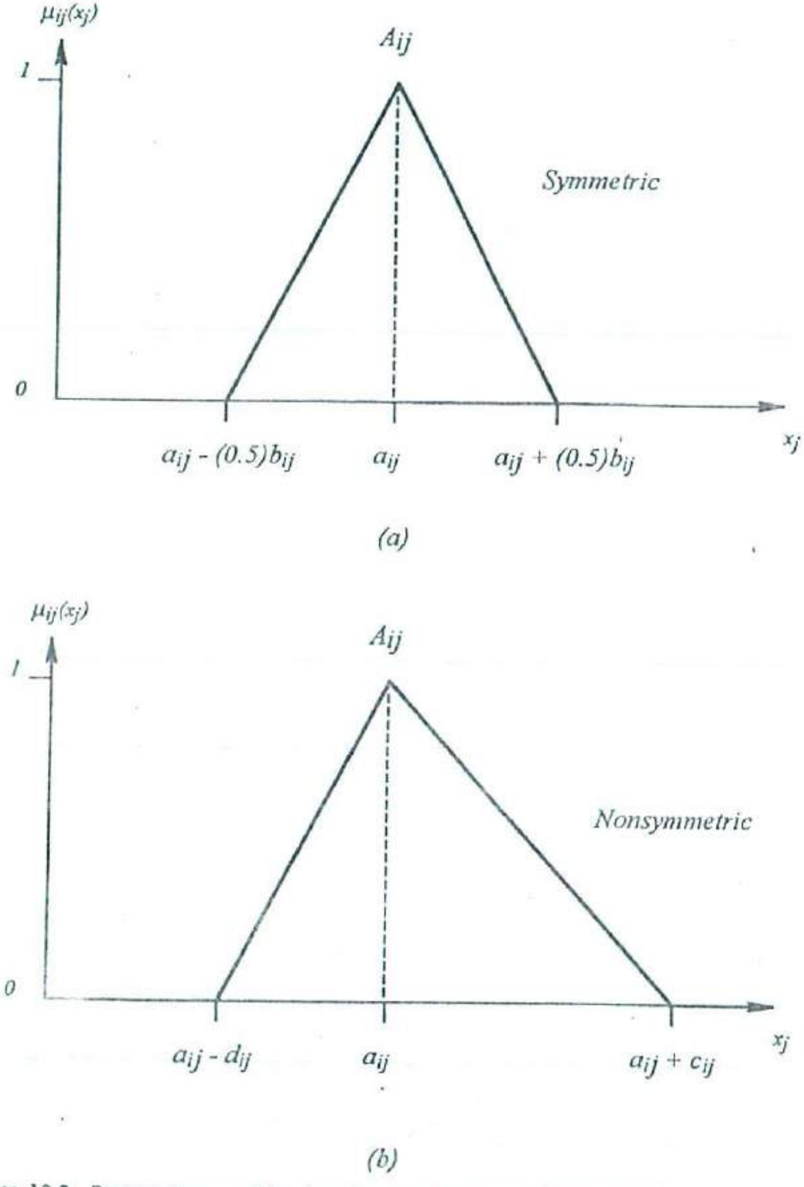
$$\begin{aligned} a_{ij}(t'+1) &= a_{ij}(t') - \frac{\eta_a}{p} \cdot \frac{\partial E}{\partial a_{ij}} \\ &= a_{ij}(t') - \frac{\eta_a}{2p} \cdot \sum_{p'=1}^{p} \frac{\partial E_{p'}}{\partial a_{ij}} \end{aligned} \qquad (13.5\text{-}9)$$

where η_b is the gradient-descent speed for a_{ij}, the peak parameter, and we use η_b and η_w for the support b_{ij} and the RHS parameter w_i, respectively. Guély and Siarry observed experimentally that learning was sensitive to these parameters.

To simplify (13.5-9) let us use E, y, and y' instead of E_p, y^p, and y'^p for the pth input–output example. Then we have the following derivative that we could use in equation (13.5-9) to obtain the update of the peak parameters:

$$\frac{\partial E}{\partial a_{ij}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial \mu_i} \cdot \frac{\partial \mu_i}{\partial \mu_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial a_{ij}} \qquad (13.5\text{-}10)$$

Similarly for the upgrade of the support and RHS parameter, we need to



(a)

(b)

Figure 13.8 Parameters used to describe the triangular-shaped membership function for the *j*th antecedent of the *i*th rule. (*a*) Symmetric triangular membership functions and (*b*) nonsymmetric triangular membership functions.

evaluate the following derivatives:

$$\frac{\partial E}{\partial b_{ij}} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial \mu_i} \cdot \frac{\partial \mu_i}{\partial \mu_{ij}} \cdot \frac{\partial \mu_{ij}}{\partial b_{ij}} \tag{13.5-11}$$

and

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_i} \tag{13.5-12}$$

Given the symmetric triangular shape of the membership functions [equation (13.5-8)] and the product interpretation of *AND*, the partial derivatives in the above equations are as follows:

$$\frac{\partial E}{\partial y} = y - y' \tag{13.5-13}$$

The partial $\partial y / \partial w_i$ is

$$\frac{\partial y}{\partial w_i} = \frac{\mu_i}{\sum_{i'=1}^{n} \mu_{i'}} \tag{13.5-14}$$

And we also have (treating y' as constant)

$$\frac{\partial y}{\partial \mu_i} = \frac{(w_i - \mu_i)}{\sum_{i'=1}^{n} \mu_{i'}} \tag{13.5-15}$$

$$\frac{\partial \mu_i}{\partial \mu_{ij}} = \frac{\mu_i}{\mu_{ij}(x_j)} \tag{13.5-16}$$

$$\frac{\partial \mu_{ij}}{\partial a_{ij}} = \frac{2\, sign(x_j - a_{ij})}{b_{ij}} \tag{13.5-17}$$

$$\frac{\partial \mu_{ij}}{\partial b_{ij}} = \frac{1 - \mu_{ij}(x_j)}{b_{ij}} \tag{13.5-18}$$

Equations (13.5-10) to (13.5-18) provide all the terms needed for the learning formula (13.5-9) for the peak, but also for the support and RHS parameters —that is, the entire set of parameters (a_{ij}, b_{ij}, w_i) we use to adapt our fuzzy systems. In general, the system is sensitive to the gradient-descent speeds, and increasing the number of rules makes training more difficult.

Similarly we can train a fuzzy system that uses nonsymmetric antecedent membership functions such as the one shown in Figure 13.8*b* by repeating the above procedure for the set of relevant parameters. Variations of the gradient-descent procedure have been developed and applied when the rules use the min interpretation of *AND* (instead of product) and a polynomial RHS instead of constant [see Guély and Siarry, (1993)]. In general, researchers report considerable advantages in the speed of training adaptive fuzzy systems when compared to regular three-layer neural networks.

13.6 ADAPTIVE NETWORK-BASED FUZZY INFERENCE SYSTEMS

To tackle the problem of parameter identification, Jang and Sun (Jang, 1992; Jang and Sun, 1995; (Jang and Gulley, 1995) have proposed an *adaptive network-based fuzzy inference system* (ANFIS) that identifies a set of parameters through a hybrid learning rule combining the backpropagation gradient-descent and a least-squares method. ANFIS can be built through the fuzzy toolbox available for MATLAB [actually developed by Jang (Jang and Gulley, 1995)]. Applications and properties of ANFIS have been investigated, and a number of methods has been proposed for partitioning the input space and hence address the structure identification problem. Fundamentally, ANFIS is a graphical network representation of Sugeno-type fuzzy systems, endowed with neural learning capabilities. The network is comprised of nodes and with specific functions, or duties, collected in layers with specific functions. To illustrate its representational strength, let us consider two first-order Sugeno rules having outputs which are linear combinations of their inputs:

$$\begin{aligned}&\text{if } x \text{ is } A_1 \ AND\, y \text{ is } B_1, \text{then } f_1 = p_1 x + q_1 y + r_1\\&\text{if } x \text{ is } A_2 \ AND\, y \text{ is } B_2, \text{then } f_2 = p_2 x + q_2 y + r_2\end{aligned} \tag{13.6-1}$$

ANFIS can construct a network realization of rules (13.6-1). Figure 13.9 illustrates the evaluation of these rules (upper part) and the corresponding ANFIS architecture (lower part). The nodes in the same layer of ANFIS are of the same function family and are arranged as follows:

Layer 1. Each node in this layer generates the membership grades of a linguistic label. The *i*th node for example may perform the following (fuzzification) operation:

$$O_i^1 = \mu_A(x) = \frac{1}{1 + \left[\left(\frac{x - c_i}{a_i}\right)^2\right]^{b_i}} \tag{13.6-2}$$

where x is the input to the *i*th node and A_i is the linguistic value (small, large, etc.) associated with this node. The set of parameters $\{a_i, b_i, c_i\}$ is used to adjust the shape of the membership function.

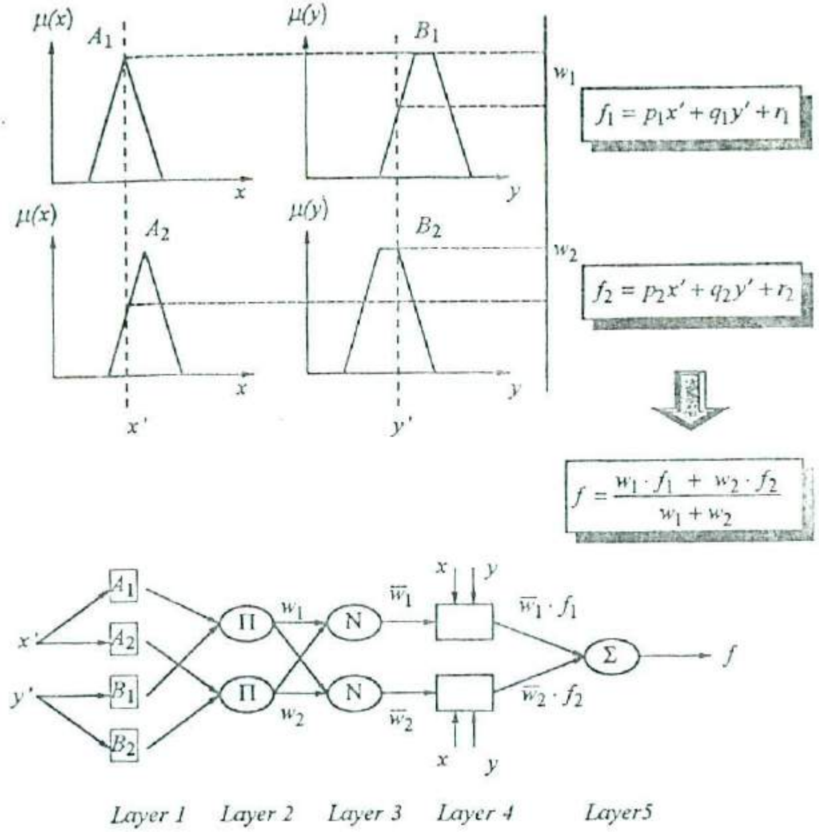


Figure 13.9 Evaluation of a network realization of rules (13.6-1) (*top*) and the corresponding ANFIS architecture (*bottom*).

Layer 2. Each node in this layer calculates the firing strength of each rule via multiplication (or min):

$$O_i^2 = w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), \qquad i = 1,2 \tag{13.6-3}$$

Layer 3. The ith node of this layer calculates the ratio of the ith rule's firing strength to the sum of all rules' firing strengths:

$$O_i^3 = \overline{w}_i = \frac{w_i}{w_1 + w_2}, \qquad i = 1,2 \tag{13.6-4}$$

Layer 4. The ith node in this layer has the following function:

$$O_i^2 = \overline{w}_i f_i = \overline{w}_i(p_i x + q_i y + r_i) \tag{13.6-5}$$

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

Layer 5. The single node in this layer aggregates the overall output as the summation of all incoming signals:

$$O_1^5 = \text{overall output} = \sum_i \overline{w}_i f_i = \frac{\Sigma_i w_i f_i}{\Sigma_i w_i} \tag{13.6-6}$$

The learning rule of ANFIS is based on gradient descent optimization as with the feedforward neural networks that we have seen in Chapters 8 and 9 [see MATLAB's fuzzy toolbox for more details on ANFIS (Jang and Gulley, 1995)].

REFERENCES

Adeli, H., and Hung S.-L., *Machine Learning—Neural Networks, Genetic Algorithms and Fuzzy Systems*, John Wiley & Sons, New York, 1995.

Blanco, A., Delgado, M., and Ruquena, I., A Learning Procedure to Identify Weighted Rules by Neural Networks, *Fuzzy Sets and Systems*, Vol. 69, pp. 29–36, 1995.

Buckley, J. J., and Hayashi, Y., Hybrid Neural Nets Can Be Fuzzy Controllers and Fuzzy Expert Systems, *Fuzzy Sets and Systems*, Vol. 60, No. 2, pp. 135–142, 1993.

Guély, F., and Siarry, P., Gradient Descent Method for Optimizing Various Fuzzy Rules, in *Proceedings of the Second IEEE International Conference on Fuzzy Systems*, San Francisco, 1993, pp. 1241–1246.

Hayashi, I., Nomura, H., Yamasaki, H., and Wakami, N., Construction of Fuzzy Inference Rules by NDF and NDFL, *International Journal of Approximate Reasoning*, Vol. 6, pp. 241–266, 1992.

Higgins, C. M., Goodman, R. M., Fuzzy Rule-Based Networks for Control, *IEEE Transactions on Fuzzy Systems*, Vol. 2, No. 1, pp. 82–88, 1994.

Ichihashi, H., Miyoshi, T., and Nagasaka, K., Computed Tomography by Neuro-fuzzy Inversion, in *Proceedings of 1993 International Joint Conference on Neural Networks*, Part 1 (of 3), Nagoya, October 25–29, 1993, pp. 709–712.

Jang, J.-S., Self-Learning Fuzzy Controller Based Inference Systems, *IEEE Transactions on Nueral Networks*, Vol. 3, No. 5, pp. 714–723, 1992.

Jang, J.-S., and Gulley, N., *Fuzzy Logic Toolbox for Use with MATLAB*, The MathWorks, Inc., Natick, MA, 1995.

Jang, J.-S., and Sun C.-T., Neuro-fuzzy Modeling and Control, *Proceedings of the IEEE*, Vol. 83, No. 3, March 1995, pp. 378–406.

Keller, J. M., and Tahani, H., Implementation of Conjunctive and Disjunctive Fuzzy Logic Rules with Neural Networks, *International Journal of Approximate Reasoning*, Vol. 6, pp. 221–240, 1992.

Keller, J. M., Hayashi, Y., and Chen, Z., "Additive Hybrid Networks for Fuzzy Logic, *Fuzzy Sets and Systems*, Vol. 66, No. 3, pp. 307–313, 1994.

Khan, E., NeuFuz: An Intelligent Combination of Fuzzy Logic with Neural Nets, in *Proceedings of the International Joint Conference on Neural Networks*, Vol. 3, 1993, pp. 2945–2950.

Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

Kuo, R. J., Cohen, P. H., and Kumara, S. R. T., Neural Network Driven Fuzzy Inference System, in *IEEE International Conference on Neural Networks—Conference Proceedings* 3, IEEE Piscataway, NJ, 1994, pp. 1532–1536.

Li, W., and Wu, Z., Self-Organizing Fuzzy Controller Using Neural Networks, in *Computers in Engineering*, Proceedings of the International Conference and Exhibit, Vol. 2, ASME, New York, 1994, pp. 807–812.

Nicholls, J. G., Martin, A. R., and Wallace, B. G., *From Neuron to Brain: A Cellular and Molecular Approach to the Function of the Nervous System*, third edition, Sinauer Associates, Sunderland, MA, 1992.

Nie, J., Neural Approach to Fuzzy Modeling, in *Proceedings of the American Control Conference*, Vol. 2, American Automatic Control Council, Green Valley, AZ, 1994, pp. 2139–2143.

Nomura, H., Hayashi, I., and Wakami, N., A Self-Tuning Method of Fuzzy Reasoning by Genetic Algorithms, in *Fuzzy Control Systems*, A Kandel and G. Langholz, eds., CRC Press, Boca Raton, FL, 1994, pp. 338–354.

O'inca Design Framework, *User's Manual*, Intelligent Machines, Sunnyvale, CA, 1994, pp. 727–748.

Pham, D. T., Bayro-Corrochano, E. J., Self-Organizing Neural-Network-Based Pattern Clustering Method with Fuzzy Outputs, *Pattern Recognition*, Vol. 27, No. 8, pp. 1103–1110, 1994.

Purushothaman, G., and Karayiannis, N. B., "Feed-Forward Neural Architectures for Membership Estimation and Fuzzy Classification, in *Proceedings of the Artificial Neural Networks in Engineering Conference* (*ANNIE '94*), St. Louis, MO, November 13–16, 1994.

Takagi, H., and Hayashi, I., NN-Driven Fuzzy Reasoning, *International Journal of Approximate Reasoning*, Vol. 5, No. 3, pp. 191–212, 1991.

Travis, M., and Tsoukalas, L. H., Application of Fuzzy Logic Membership Functions to Neural Network Data Representation, Internal Report, University of Tennessee, Knoxville, TN, 1992. Included as Appendix A in Uhrig et al., *Application of Neural Networks*, EPRI Report TR-103443-P1-2, January 1994.

Uhrig, R. E., Tsoukalas, L. H., and Ikonomopoulos, A., Application of Neural Networks and Fuzzy Systems to Power Plants, in *Proceedings of the 1994 IEEE International Conference on Neural Networks*, Vol. 2, Part 6 (or 7), Orlando, FL, June 27–29, 1994, pp. 510–512.

Wang, L.-X., *Adaptive Fuzzy Systems and Control*, Prentice-Hall, Englewood Cliffs, NJ, 1994.

Wang, L. X., and Mendel, J. M., Backpropagation Fuzzy Systems as Nonlinear Dynamic System Identifiers, in *IEEE International Conference on Fuzzy Systems*, San Diego, CA, 1992, pp. 1409–1418.

Werbos, P. J., Neurocontrol and Fuzzy Logic: Connections and Designs, *International Journal of Approximate Reasoning*, Vol. 6, pp. 185–219, 1992.

Yager, R. R., Modeling and Formulating Fuzzy Knowledge Bases Using Neural networks," *Neural Networks*, Vol. 7, No. 8, pp. 1273–1283, 1994.

Yea, B., Konishi, R., Osaki, T., and Sugahara, K., Discrimination of Many Kinds of Odor Species Using Fuzzy Reasoning and Neural Networks, *Sensors and Actuators, A: Physical*, Vol. 45, No. 2, pp. 159–165, 1994.

PROBLEMS

1. Show a nontriangular form for the membership function of the ith instance in the p cluster described by Equation 13.3-7 and discuss its potential benefit.

2. Derive Equation (13.4-6) in the Takagi-Hayashi (T-H) method.

3. Explain qualitatively the significance and use of Equation (13.4-7).

4. Explain qualitatively how the number of variables in the consequent of Equation (13.4-1) is controlled in the T-H method.

5. Show that for a fuzzy algorithm comprised of zero-order Sugeno rules the output is given by Equation (13.5-4). Identify the parameters that may be used for training.

6. Using input–output data and gradient descent we can modify (adapt) the parameters of a fuzzy algorithm in a manner analogous to neural learning. Show how the equation for updating parameters such as weights, that is Equation (13.5-7), is obtained.

7. Given a fuzzy algorithm comprised of n first-order Sugeno rules, derive expressions analogous to Equations (13.5-3) and (13.5-4) and identify all parameters that may be used for training.

8. How can the parameters of the fuzzy algorithm of Problem 7 be trained? Describe all assumptions that need to be made and give the learning rule for each parameter.

9. If min instead of product is used in fuzzy algorithms comprised of zero-order Sugeno rules, how would their parameter training be different?

10. Derive expressions for the training parameters involved in fuzzy algorithms that use nonsymmetric triangular membership functions such as shown in Figure 13.8.

14

SELECTED HYBRID NEUROFUZZY APPLICATIONS

14.1 INTRODUCTION

Recent years have seen a rapidly growing number in neural-fuzzy applications and a blossoming bibliography on the subject.[1] Although it is too early for the merits of any particular approach to be comprehensively assessed, it appears that in a number of engineering disciplines the research is maturing and moving toward developmental phases. In this chapter we describe selected hybrid neurofuzzy engineering applications. The task of reporting on a field that is still in a state of flux is difficult and tricky, and unfortunately our selection is incomplete. Nevertheless, we think it may be useful to offer a panoramic view of applications through the neurofuzzy bibliography.

In Part II of this book we have seen that problems associated with obtaining expert knowledge and adapting a system description to changes in itself or its environment can be addressed through neural networks. Since neural descriptions of systems are typically made through example data or some kind of performance function, expert knowledge is not explicitly required. In addition, neural networks are inherently capable of adaptation through the various learning algorithms which were reviewed earlier. It would seem plausible, therefore, to try to overcome the expert knowledge and adaptation problems of fuzzy systems through synergistically exploiting these advantageous features of neural networks.

[1] The material presented in this chapter is largely a condensation of research reports in neurofuzzy applications that have appeared in the early 1990's, including material obtained through searches at Purdue University Library's Engineering Index.

Table 14.1 Comparative characteristics of fuzzy and neural systems

Fuzzy Systems	Neural Systems
Linguistic Representation	*Black Box Representation*
Expert Knowledge Required	*Example Data or Performance Function Required*
Some Adaptation	*Adaptation Mechanisms Available*
Fault Tolerant	*Fault Tolerant*
Application-Dependent Computational Cost	*Rather High Computational Cost*
Multiple Descriptions Possible	*Multiple Descriptions Possible*

Neural networks exhibit highly desirable inherent parallelism and fault-tolerant behavior. Of course, they have disadvantages of their own such as, for example, difficulties in inspecting and modifying internal parameters. Whereas fuzzy systems are relatively easy to inspect and modify, neural networks are not as transparent to a user. In addition, there may be situations where adequate data are simply not readily available, which could cause difficulties in training or possibly a high computational cost associated with training. Table 14.1 presents a comparison and a summary of the characteristics of fuzzy and neural systems.

14.2 NEUROFUZZY INTERPOLATION

The notion of interpolation typically refers to a process whereby we estimate the value of a function between values that are already known. More generally, this notion refers to methods for approximating a function with a simpler one, when interpolating values or derivative values are provided, as is the case in spline fitting of Langrange interpolation.

In fuzzy logic, we deal primarily with complex many-to-many mappings rather than the simple many-to-one mappings (or functions). Consider the situation shown in Figure 14.1*a*, where the empty circles represent known fuzzy rules (see Chapter 5). As seen in the figure, there are regions where such knowledge (i.e., the underlying relations) is missing. We can think of this problem in a manner analogous to crisp interpolation, that is, find a

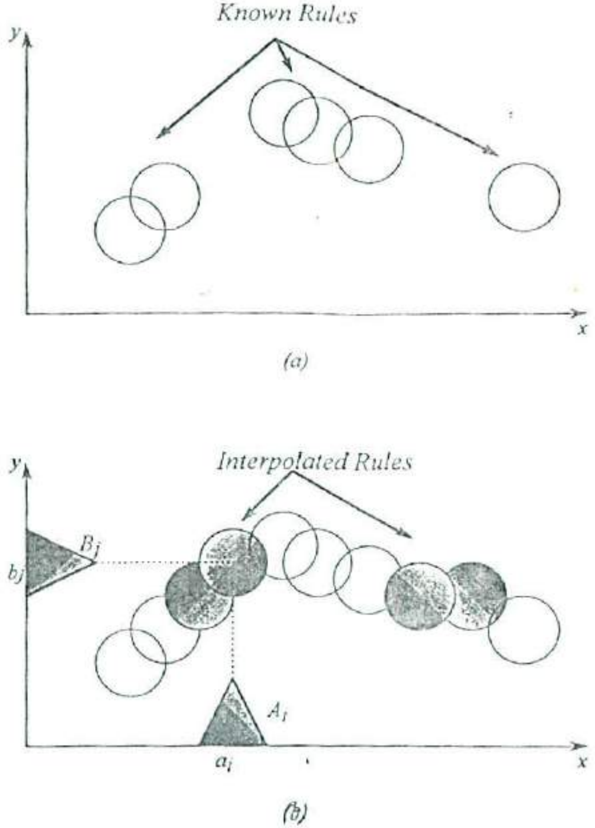


Figure 14.1 Neurofuzzy interpolation involves the use of neural methods for obtaining the *interpolated rules.*

method through which we can estimate the relation in the missing part. When this is accomplished through neural methods, we have what is known as *neurofuzzy interpolation*. To obtain the rules involves finding the appropriate membership functions as shown schematically in Figure 14.1*b* where the shaded circles represent interpolated rules.

Abe and his colleagues at Hitachi (Abe and Lan, 1993) have developed a method for extracting fuzzy rules directly from numerical input–output data for pattern classification in a manner similar to neural networks and extended it to approximate any arbitrary function. For function approximation, the universe of discourse of an output variable is divided into multiple intervals, and each interval is treated as a class. Then in a manner similar to that used for pattern classification, fuzzy rules are recursively defined by

(a) *activation hyperboxes* which show the existence region of the data for the interval and (b) *inhibition hyperboxes* which inhibit the existence region of data for that interval. Input data are used for each individual interval. The approximation accuracy of the fuzzy system derived by this method has been empirically studied by Abe and Lan (1993) using an operation learning application of a water purification plant and found to be satisfactory. Additionally, it has been reported that the approximation performance of the fuzzy system compares favorably with the function approximation approach based on neural networks.

Blanco and Delgado (1993) have also developed an interpolation method based on a neural network's ability to approximate any function. The methodology involves a neural network learning the information contained in fuzzy rules, as well as expert knowledge found in a set of examples, and directly interpolating from rules through the output of neural networks.

Kosko (1994) has shown that an additive fuzzy system can uniformly approximate any real continuous function on a compact domain to any degree of accuracy. An additive fuzzy system approximates the function by covering its graph with fuzzy patches in the input–output state space and averaging patches that overlap. The fuzzy system computes a conditional expectation $E[Y|X]$ if the fuzzy sets are viewed as random sets. Each fuzzy rule defines a fuzzy patch and utilizes common-sense knowledge with state-space geometry. Neural or statistical clustering systems can approximate the unknown fuzzy patches from training data. Kosko (1994) has reported that these adaptive fuzzy systems approximate a function at two levels. At the local level the neural system approximates and tunes the fuzzy rules. At the global level the rules or patches approximate the function.

14.3 GENERAL NEUROFUZZY METHODOLOGICAL DEVELOPMENTS

Leading a research field from infancy to maturity and technological deployment is a particularly difficult task, and crucial methodological developments obtained through the insight and intuition of experienced researchers make a difference. Let us take a look at some of these pivotal methodological advancements which have contributed to neurofuzzy integration.

Werbos (1993) introduced the concept of *elastic fuzzy logic* as a way of combining neural and fuzzy capabilities. Werbos' methodology uses fuzzy logic as a kind of "translation" technology, to go back and forth between the words of a human expert and the equations of a controller, a classifier, or some other useful system. One can then use neural methods to adapt that system to improve performance. Elastic fuzzy logic translates the words of an expert into an elastic fuzzy logic network, a kind of local neural network which can be plugged directly into a wide range of neural network designs, ranging from pattern classification through the brain-like optimizing control. The words of the expert are used to initialize this network, but neural network methods can then be used to adapt all the weights or parameters. In Werbos' methodology, neural network methods can also be used to prune or grow the network.

Ronald Yager, a prominent researcher in the field of fuzzy systems, has advanced a general framework for developing fuzzy algorithms using neural networks. Yager (1994) interprets the firing level of a neuron as a measure of possibility between two fuzzy sets, the weights of connection and the input, and suggests a way to represent fuzzy production rules in a neural framework. Central to Yager's representation is the notion that the linguistic variables associated with a fuzzy *if/then* rule may be represented as weights in the resulting neural structure. Such a structure allows for learning of the membership functions involved.

Several investigators have proposed neural-network-based fuzzy systems. A leading part of the research and important methodological advancements have come out of the work of Professor Keller and his coworkers at the University of Missouri (Keller et al., 1994; Keller and Tahani, 1992). Over the years they have developed a variety of approaches toward improving the performance of various systems by exploiting the neurofuzzy synergism. They have introduced evidence aggregation networks based on additive fuzzy hybrid operators, for image segmentation, pattern recognition, and general multicriteria decision-making. These networks have excellent properties for decision-making under uncertainty and present advantages in training due to their simple form. Keller's additive hybrid operators are found to be flexible and useful for modeling nodes in a network structure for fuzzy logic inference capable of learning appropriate functional relationships while being rather transparent; that is, after training, individual nodes can be analyzed as a collection of "mini-rules".

Neural networks for the parallel high-speed processing of the rules found in a fuzzy logic controller have been used by Patrikar and Provence (1993) at Southern Methodist University. In the methodology advanced by the researchers, the fuzzy algorithm is replaced by a feedforward neural network with a single hidden layer that is trained using backpropagation and input and output fuzzy values expressed in terms of numerical patterns.

As we have seen in Chapter 13, the automatic categorization of a universe of discourse is typically based on some type of Kohonen network. A pattern clustering method based on the Kohonen feature mapping algorithm and the backpropagation multilayer perceptron has been used for membership function determination by Pham and Bayro-Corrochano (1994). The method is applied first to the training data set to divide it into labeled clusters using the Kohonen algorithm and a simple cluster labeling procedure. The data clusters are then employed to train a three-layer perceptron using the error backpropagation training. Thus, this approach is self-organizing by virtue of the Kohonen algorithm and produces fuzzy outputs as a consequence of the backpropagation network. Results of using the pattern clustering method on standard problems show it to be superior in performance compared to crisp clustering networks such as the Kohonen feature map and the ART-2 network [see also Nie (1994)].

Feedforward neural estimation for membership function determination and fuzzy classification has been proposed by Purushotaman and Karayiannis (1994) at the University of Houston. They have used feedforward neural networks inherently capable of fuzzy classification of overlapping pattern classes such as a feedforward neural network in conjunction with multilevel neurons in two hidden layers called (a) the "quantum neural network" and (b) a "membership estimating network," which is a feedforward network trained with generalized Hebbian learning rules. Professor Karayiannis and his students have offered theoretical and experimental results showing that both architectures are inherently capable of partitioning the feature space in a fuzzy manner.

To a large extent, the successful implementation of neural nets depends on several ancillary techniques for data preprocessing, training, and testing. Some of these techniques were investigated and discussed by Professor El-Sharkawi of the University of Washington [see El-Sharkawi (1994)]. They include genetic algorithms, fuzzy logic theory, query-based learning, and feature extraction. The advantages of the application of these ancillary techniques for neural networks and simulation studies have been performed to assess their role and practicality.

14.4 ENGINEERING APPLICATIONS

Fuzzy and neural approaches have found their way in a variety of engineering applications, including, but not limited to, consumer electronics, various aspects of control, diagnostics, industrial production lines, biotechnology, power generation, chemical processes, power electronics, communications, and software resource management. It is expected that the applications of the fuzzy-neural synergism will increasingly move toward computer applications as well, such as machine learning [see Adeli and Hung (1995)].

It is now rather well established that fuzzy systems aided by neural networks can adequately address the adaptation problems we discussed in Chapters 12 and 13. Ishibuchi and his coworkers have reported on an approach based on empirical research where they examine the ability of trainable fuzzy systems as approximators of nonlinear mappings by computer simulations using real-life data. Fuzzy *if/then* rules of the Sugeno variety (see Chapter 6) are adjusted by a gradient descent method. After examining the capabilities of fuzzy systems by numerical examples, the researchers tested them through an interesting project, involving the development of a six-variable fuzzy relation used in rice tasting. By computer simulations based on a random subsampling technique, they demonstrated that the performance of individual fuzzy systems is comparable to that of neural networks (Ishibuchi et al., 1994).

Researchers at Tohoku University in Japan have used neural networks in conjunction with a fuzzy logic for decision-making (Kozma et al., 1994). They developed a method which can make a distinction between the occurrence of unexperienced events and any inconsistency in the judgments of agents caused by statistical uncertainties in actual data. The method has been applied to the analysis of signals of numerical experiments and also actual measurements in a nuclear reactor.

Several fuzzy-neural methodologies are of special interest to many researchers when integrated with other approaches. Pao (1994) reported on the fusion of three distinct computational intelligence paradigms, neural computing, evolutionary programming, and fuzzy-logic, to support the task of process monitoring and optimization. The resulting computational intelligence has been successfully applied to optimal process planning in electric power utilities that include, but are not limited to, heat rate improvement and NO_x emission minimization.

14.5 DIAGNOSTICS IN COMPLEX SYSTEMS

Hybrid fuzzy-neural systems have been used in several aerospace applications. Raza, Ioannou, and Youssef (1994) reported on the problem of detecting control surface failures of a high-performance aircraft. The detection model is developed using a linear, six-degree-of-freedom dynamic model of an F-18 aircraft. The detection scheme makes use of a residual tracking error between the actual system and the model output in order to detect and identify a particular fault. Two parallel models detect the existence of a surface failure, whereas the isolation and magnitude of any one of the possible failure modes is estimated by a decision algorithm using neural networks and fuzzy logic. Simulation results demonstrate that detection can be achieved without false alarms even in the presence of actuator/sensor dynamics and noise.

In the power industry, neural networks and fuzzy logic systems offer an interesting, challenging, and productive means of addressing many of the problems that occur in the operation of nuclear power plants. Uhrig, Tsoukalas, and Ikonomopoulos (1994) have described how such systems can be used to model nuclear reactor system dynamics and nuclear fission step responses of nuclear plants. They can also help operators in assessing the condition of the plant during abnormal operation or emergencies by analyzing and integrating the process parameters and system interactions (Guo and Uhrig, 1992).

Matsuoka and Blanco (1993) reported on an Electric Power Research Institute (EPRI) survey of recent advances and trends in Japanese power plants. The survey includes case studies of many applications and widespread implementations of advanced technologies such as: (1) a neurofuzzy system for plant monitoring and diagnosis, combined with knowledge-based preventive maintenance systems: (2) fuzzy-logic dynamics schedulers for plant transient operations; (3) fuzzy-expert tuners of dynamic control systems; (4) fuzzy-algorithmic operation guidance systems for major plant equipment; and (5) telepresence with machine vision and robotics; among others. These advanced approaches had to be introduced due to smaller stability margins in the plants, rapid changes toward more efficient thermal cycles and new plant equipment dynamics, coming with stronger nonlinearities and subsystem interconnections.

In the field of nuclear engineering diagnostics is a very important task for the safety of power plants. Moon and his coworkers at the Korean Advanced Institute on Science and Technology have reported on a method for predicting the critical heat flux (CHF)—a quantity with safety significance— based on fuzzy clustering and neural networks [Moon and Chang, (1994)]. The fuzzy clustering classifies the experimental CHF data into a few data clusters (data groups) according to the data characteristics. After classification of the experimental data, the characteristics of the resulting clusters were carefully examined. Using the CHF data in each group, neural networks were trained and successfully predicted the CHF.

14.6 NEUROFUZZY CONTROL SYSTEMS

In another application, Chen and Chen (1994) have investigated the relationship between a piecewise linear fuzzy controller (PLFC), in which the membership functions for fuzzy values and the fuzzy *if/then* rules are all in piecewise linear forms, and a Gaussian potential function network-based controller (GPFNC), in which the network output is a weighted summation of hidden responses from a series of Gaussian potential function units. Systematic procedures were developed for transformations from a PLFC to its GPFNC counterpart, and vice versa. Based on these transformation principles, a series of systematic and feasible steps were developed for the design of an optimized PLFC (PLFC*) using neural network techniques. The optimized GPFNC (GPFNC*) can be implemented directly to actual systems, and the GPFNC* could further be converted into its fuzzy counterpart (PLFC*) if more structural interpretation of the intelligent control strategy is required.

Several self-organizing fuzzy controllers have found their way to field deployment. Li and Wu (1994) developed an interesting a self-organizing fuzzy logic control scheme based on neural networks, which consists of a traditional fuzzy logic controller and a conventional derivative controller.

Neural networks are used to optimize membership functions that are parameterized by the use of the cubic splines in a self-organizing manner.

In another development, Professors Lin and Lee (1994) have proposed a promising approach for constructing a fuzzy system automatically. In their approach a reinforcement neurofuzzy control system with multiple connectionist models with feedforward multilayered networks is used to realize a fuzzy logic controller. One network performs the role of a fuzzy predictor, while the other acts as a fuzzy controller. Using the temporal difference prediction method, the fuzzy predictor can predict the external reinforcement signal and at the same time provide a more informative internal reinforcement signal to the fuzzy controller. During the learning process, both structure learning and parameter learning are performed simultaneously in the two networks using a fuzzy similarity measure, and a reward/penalty signal.

As far as the practical implementation of neurofuzzy control is concerned, there is tremendous variation in the themes and areas of applications. Stylios and Sotomi (1994) have developed a neurofuzzy sewing controller for the next generation of, the so-called intelligent sewing machines. The model incorporates discrimination of material characteristics to be stitched and automatic determination of their properties. The fabric–machine interactions at different speeds have been articulated in the form of fuzzy *if/then* and implemented in a neural network to allow for optimization of fuzzy membership functions and, subsequently, self-learning. The controller was successfully applied to an instrumented industrial sewing machine.

Neurofuzzy approaches are expected to play a major role in the development of future fusion reactors. Yamazaki et al. (1994) reported that the world's largest superconducting fusion machine LHD (large helical device), under construction in Japan, will utilize fuzzy logic and neural networks for feedback control of plasma configurations in addition to classical proportional-integral-derivative control. Design studies of the control system and related R & D programs with coil-plasma simulation systems include neurofuzzy control systems.

Foslien and Samad (1993) at Honeywell reported on the general problem of optimizing a fuzzy controller through the use of a neural network model for the process in the optimization procedure. The integration of nĕural network models with fuzzy control is very appropriate since both techniques are best used when detailed analytical understanding of a process is not available. To illustrate this concept, a fuzzy controller was synthesized for a simple nonlinear process with (1) a feedforward neural network used for modeling the process and (2) an optimization criterion based on setpoint error.

The synergistic utilization of fuzzy and neural systems, often resulting in an entity of its own referred to as *neurofuzzy systems*, is increasingly applicable in many control technologies (Werbos, 1992). As we have seen in Chapter

6, for example, a di[illegible]lt part in designing an ordinary fuzzy controller is selecting which fuzzy [illegible] are best representing the controlled and controlling variables. Most fuzzy [illegible]ntrollers are sensitive to the shapes of the membership functions, and [illegible]he number of rules increases, the use of "trial and error" tuning proced[illegible]s become less and less feasible.

A report in *IEEE* [illegible]*ctrum* magazine (Schwartz and Klir, 1992) described work at Matsushita [illegible]d Hitachi in Japanese in which a backpropagation neural network learne[illegible] the needed membership functions from a set of training examples (Hayashi et al., 1992). It is claimed that a tuning task that had previously taken 6 months was accomplished in 1 month. Wakami et al. (1993) at Matsushita Electric reported on recent applications of fuzzy-neural methodologies to home electric appliances. Many appliances produced in Japan have internally encoded expert knowledge for their operation. In order to overcome the problem of extracting the necessary expertise, Matsushita engineers use neural networks in conjunction with fuzzy rules. Applications of their neurofuzzy methods are found in refrigerators, air-conditioning systems, and welding machines. In air-conditioning systems, a thermal sensory system and a fuzzy-image-understanding algorithm are used to identify the number and positions of occupants in a room. Allowing air-conditioning systems to "see" their environment allows them to better and more efficiently produce a comfortable thermal environment.

As far as the industrial merit of neurofuzzy technologies is concerned, Wegmann (1994) at Siemens reported a growing interest in programmable logic controller (PLC) applications and a wide range of possible applications in the area of nonlinear processes, especially those with great parameter fluctuations. Applications in environmental processes, such as sewage and exhaust gas cleaning, appear to be of particular interest. Neural networks alone may be at a disadvantage in operating phases of a process where example data are not readily available, whereas neurofuzzy formulations lend themselves conveniently to such situations allowing the control behavior in such phases to be prescribed by a fuzzy algorithm, with most learning left to neural networks.

In another interesting application, Yen (1994) reported on the design of control algorithms for flexible space structures, possessing nonlinear dynamics which are often time-varying and usually ill-modeled. A hybrid connectionist system was used as a learning controller with reconfiguration capability. Neural networks were used to provide vibration suppression and trajectory maneuvering for precision pointing of flexible structures. Radial basis function networks were employed for capturing spatiotemporal interactions among the structure members. A fuzzy-based fault diagnosis system provided the neural controller with various failure scenarios, and the associative memory incorporated into the adaptive architecture compensated for catastrophic changes of structural parameters by offering a continuous solution space of acceptable controller configurations.

Sharaf and Lie (1994) reported on a novel neurofuzzy hybrid power system stabilizer designed for damping electromechanical modes of oscillation and enhancing power system synchronous stability. The hybrid system comprises a front-end conventional analog power system stabilizer design, an artificial neural network based stabilizer, and a fuzzy logic postprocessor gain scheduler.

In the power electronics field, Professor Bose at the University of Tennessee reported on new applications emerging in the field that exploit fuzzy and neural approaches along with other AI techniques (Bose, 1994).

14.7 NEUROFUZZY CONTROL IN ROBOTICS

In the field of robotics, there is a booming interest in neurofuzzy means for supervisory control, planning, grasping, and guidance, and a variety of applications are found (Kuo, 1993; Kuo et al., 1994). Professor Bourbakis and his colleagues at Binghamton University (Tascillo et al., 1993) have developed a neurofuzzy hand-grasp algorithm for improving the first grasp of a wheelchair robotic arm with two three-joined fingers and a two-jointed thumb. The robotic arm uses pressure and force feedback and a learning mechanism that helps to avoid an extensive search of an optimal grasp each time an object is lifted.

Hanes et al. (1994) reported on an intelligent control architecture for a robotic grasping system capable of acquiring an object into a fully enveloping power grasp. Control of the internal forces of the grasp is provided, along with trajectory control of object position, as the object is picked up. Fuzzy control techniques are used for control of internal forces in the power grasp, and a neural network provides a means of in-process nonlinear friction estimation.

Fatikow and Wohlke (1994) reported on a neurofuzzy architecture for the intelligent control of multifinger robot hands. The control system is based on the combination of a neural network approach for the adaptation of grasp parameters and a fuzzy logic approach for the correction of parameter values given to a conventional controller. A planning component of the system determines initial manipulation parameters, while a neural network performs continual computations of suboptimal grasp forces. On-line learning of fuzzy *if/then* rules is used for parameter adjusting.

A neurofuzzy controller for adaptive tracking in unknown nonlinear dynamic systems and for on-line computation of inverse kinematic transformations of a two-linked robot has been developed by Rao and Gupta (1994). The controller is comprised of a fuzzy algorithm in the feedback configuration and a recurrent neural network in the inverse mode (feedforward) configuration. The controller provides a means for converting a linguistic control strategy into control actions while the neural network provides sensory (low-level) computations and embodies important features such as learning, fault-tolerance, parallelism, and generalization in a manner similar to the one we have seen in Chapter 13.

14.8 PATTERN RECOGNITION AND IMAGE ENHANCEMENT

Neural networks have been extensively used in connection with fuzzy algorithms for edge detection, and in connection with fuzzy means for defining parameters they are producing interesting realizations of neurofuzzy systems. Kim and Cho (1994) reported on an edge relaxation method utilizing fuzzy logic and neural networks where candidates for edge segments are first estimated using a local derivative operator with a window of small size. Fuzzy *if/then* rules, each of which is associated with a neighborhood pattern defined by the spatial relationships among the neighboring edge segments, are used as a computational framework of collecting the evidence for the existence of an edge segment. The fuzzy rules are trained by a specially structured neural network which performs a fuzzy reasoning operation.

Improvements on clustering algorithms are being investigated by many researchers. The extension of neural-net-based crisp clustering algorithms to fuzzy clustering algorithms has been addressed extensively. For a comprehensive review see the excellent compilation of papers in the book *Fuzzy Models for Pattern Recognition*, edited by J. C. Bezdek and A. K. Pal (Bezdek and Pal, 1994). However, many neurofuzzy clustering algorithms developed so far suffer from restrictions in identifying the actual decision boundaries among clusters with overlapping regions. These restrictions are induced by the choice of the similarity measure and the representation of clusters. An integrated adaptive fuzzy clustering algorithm was developed by Kim and Mitra (1994) to generate improved decision boundaries by introducing a new similarity measure and by integrating the advantages of the fuzzy optimization constraint of fuzzy *c*-means, the control structure of adaptive resonance theory (ART-1), and a fuzzified Kohonen-type learning rule.

Dalton (1994) at Apple Computers reported on a fuzzy-neural approach to image manipulations that allows a user to quantify qualitative aesthetics. Image enhancement and other desired manipulations are thought of as nonlinear transformations from an input space of arbitrary images into an output space of desired aesthetic images. Derivation of imaging manipulations of this type can be viewed as supervised learning problems that can be solved by neural methods. In order to reduce the dimensionality of the transformations involved, descriptors more structured than raw image pixels may be used; hence, imaging transformations between sets of image metrics as opposed to sets of image pixels can be learned by the network (from example images). Alternatively, an adaptive fuzzy algorithm can be used to achieve the underlying functional transformation while providing a link between semantic labeling of qualitative image characteristics and the underlying raw image data.

Kulkarni et al. (1994) have proposed a neural network model for fuzzy logic decisions consisting of six layers; the first three layers map the input variables to membership functions, and the last three layers implement the decision rules. Triangular membership functions are used, and the model learns the decision rules using a supervised gradient descent procedure. The connection strengths between the last three layers encode the decision-rules used in decision-making. Layer 1 is the input layer that receives the input features, while layer 2 represents the linguistic variables (with five values, *VERY LOW, LOW, MEDIUM, HIGH*, and *VERY HIGH*) for each input feature; Hence, layer 2 has five times as many nodes as layer 1. Each node of layer 2 is connected with weights ±1 to two nodes in layer 3 where the two nodes represent the left and right sides of the triangular membership functions. Each node in layer 4 combines the outputs of the corresponding two nodes in layer 3 so that it now represents the membership values, which is presented to layer 5. Layers 5 and 6 are implementing the inference process. Layers 4, 5, and 6 represent a simple three-layer feedforward network with backpropagation learning. The number of nodes in the output layer is equal to the number of output decisions. During training, only the weights between layers 4, 5, and 6 are adjusted.

The above system has been successfully used to recognize objects in multispectral satellite images based on data obtained from thematic mapper sensors (a multispectral scanner that captures data in seven spectral bands). Five inputs to layer 1 were used, and layers 2, 3, and 4 contained 25, 50, and 25 nodes, respectively, since five linguistic values were used. Layers 5 contained 35 nodes, and layer 6 contained 5 nodes representing output categories. The researchers have reported that results obtained were virtually identical with results from a three-layer conventional neural network classifier and a conventional maximum likelihood classifier. However, the conventional neural network took over 24 hours to train as opposed to about 25 minutes for the fuzzy neural system. Both the conventional and fuzzy neural network systems gave results very rapidly after training. In contrast, the conventional maximum likelihood classifier had to handle each pixel individually and sequentially; as a result, the conventional classifier took excessively long times for classification.

14.9 MEDICAL AND ENVIRONMENTAL IMAGING USING NEUROFUZZY METHODOLOGIES

The extraction of fuzzy values is of particular interest in medical imaging, where a plethora of data-rich situations exist. Computed tomography, magnetic resonance, digital ultrasound, and other forms of computer-assisted radiology provide an unprecedented volume of data that, if interpreted correctly, lead to the visualization of the body's internals and diagnosis of subtle decease processes at a very early stage of development (Ichihashi et al., 1993). Physicians and engineers collaborate in many areas of medicine to develope and use revolutionary computer-assisted techniques for education, visualization, diagnostics, and telesurgery, amongst others.

Brotherton et al. (1994) have developed a neurofuzzy system to automatically classify structures and tissues in echocardiograms. The system performs structure classification as a first step using advanced multiple-feature, hierarchical, fuzzy neural network fusion approach. It learns to classify tissue types by examination of image training data. Classification assigns each image pixel a fuzzy membership measure for each structure or tissue type. Final hard classification, if required, is delayed until the system's output stage. This allows important information to be retained throughout the system. The first layer in the hierarchy of networks determines gross spatial relationships and texture classes, while the second layer fuses the spatial and textual net outputs to make final classifications.

In a related medical imaging problem, T. Chen, W-C. Lin, and C-T. Chen (Chen et al., 1994) at Argonne National Laboratory have developed a fuzzy neural network based approach to 3-D heart motion understanding using expert cardiologist knowledge to specify different classes of motion and obtain classification rules. The objective is to find the decisions for all possible classes of motion in the form of possibilities. Experiments on real data have been conducted to corroborate the neurofuzzy approach.

On the environmental side of neurofuzzy applications, Barbosa et al. (1994) in Brazil reported on a neural system for deforestation monitoring through automatic interpretation of satellite images of the Amazon region. Their approach is based on a combination of image segmentation and classification techniques, the latter employing a neural network architecture that works on a fuzzy model of classification. It appears that such an approach has a range of advantages over more traditional, pixel-based approaches employing statistical techniques, ranging from the possibility of treating transition and interference phenomena in the images to the ease with which complex information related to a region's geometry, texture, and contextual setting can be used.

14.10 TRANSPORTATION CONTROL

In the field of transportation engineering, efforts to manage freeway congestion have been seriously impeded by the inability to promptly and reliably detect the presence of traffic incidents. Traditional incident-detection algorithms distinguish between congested and uncongested operations by comparing measured traffic-stream parameters with predefined threshold values. Given the range of possible operating conditions in a certain traffic stream, selecting a single threshold value may be a difficult and uncertain decision. A system called *fuzzy logic incident patrol system* was developed by Hsiao, Lin, and Cassidy (1994) to solve many of the problems inherent in traditional incident-detection algorithms. The *fuzzy logic incident patrol system* is a hybrid neurofuzzy system constructed from training examples to find the optimal input–output membership functions. Threshold values, implicitly obtained by *if/then* rules and membership functions, are treated as dependent variables, which change according to prevailing traffic-stream parameters measured by detectors.

14.11 ADAPTIVE FUZZY SYSTEMS

Neural-network-based adaptive fuzzy systems have been used in the field of seismic evaluation. Chu and Mendel (1994) have developed a method for solving the so-called "first break picking" problem in seismic signal processing, one that requires much human effort and is difficult to automate. The goal has been to reduce the manual effort in the picking process and accurately perform the picking. A backpropagation fuzzy logic system has been used for first break picking by employing derived seismic attributes as features. Experimental results reported by Chu and Mendel have indicated that this neurofuzzy system achieves about the same picking accuracy as a feedforward neural network that is also trained using a backpropagation algorithm; however, it is trained in a much shorter time because there is a systematic way in which initial parameters can be chosen, as opposed to the random way in which the weights of the neural network are chosen.

Mitra and Pal (1994) have proposed a self-organizing artificial neural network, based on Kohonen's model of self-organization, which is capable of handling fuzzy inputs and of providing fuzzy classification. Unlike conventional neural net models, this algorithm incorporates fuzzy set-theoretic concepts at various stages. The input vector consists of membership values for linguistic properties along with some contextual class membership information which is used during self-organization to permit efficient modeling of fuzzy (ambiguous) patterns. A new definition of gain factor for weight updating has been proposed by the researchers. Incorporation of the concept of fuzzy partitioning allows natural self-organization of the input data, especially when they have ill-defined boundaries. The output of unknown test patterns is generated in terms of class membership values. Incorporation of fuzziness in input and output is seen to provide better performance than a Kohonen model.

Fei-Yue Wang and D. D. Chen (Wang and Chen, 1994) have investigated general principles involved in the design of adaptive fuzzy controllers via neural networks and proposed a method that implements a rule-based fuzzy control system via a neural network consisting of two subnetworks: one for pattern recognition and the other for fuzzy reasoning and control synthesis. The neural network is arranged in such a way that the structure and operations of the original fuzzy control system can be fully retrieved from its network implementation. Equipped with the learning capability of neural networks, this implementation provides a mechanism for refining the existing rules and generate new rules for fuzzy control (Wang, 1994)

14.12 INSPECTION USING NEUROFUZZY METHODS

In the area of fault diagnosis, Goode and Chow (1994) presented a novel hybrid fuzzyneural fault detector that will use the learning capabilities of the neural network to detect if a motor has an incipient fault. Once the neurofuzzy fault detector is trained, heuristic knowledge about the motor and the fault detection process can also be extracted. With better understanding of heuristics through the use of fuzzy rules and fuzzy membership functions, a better understanding of the fault detection process of the system is obtained.

Moganti, Dagli, and Ercal (1994) developed a fuzzy-neural method for automatically inspecting printed circuit boards for defects. The process involves a two-level classification of the board image subpatterns into either standard nondefective patterns or defective patterns. The patterns that are identified as being defective in the first level are thoroughly checked for defects in the second level, and the patterns that are nondefective are checked for dimensional verification for the classes that a board has been identified and assigned to the correct class.

14.13 NEUROFUZZY METHODS IN FINANCIAL ENGINEERING

Another application where the use of neural network technology is introducing fuzzy concepts is in the financial community. It is well known that neural networks have been used for several years in the selection of investments because of their ability to identify patterns of behavior that are not readily available. Much of this work has been proprietary for the obvious reason that the users want to take advantage of their insight into the market gained through the use of neural network technology.

In the past year, some financial work has incorporated neurofuzzy technology. Hobbs and Bourbakis (1995) have described a neurofuzzy simulator used for stock investing that identifies patterns associated with whether a stock is underpriced or overpriced. Since stock prices are determined by what a buyer will pay, most stocks tend to be underpriced or overpriced at one time or another. Eventually, the price corrects itself, but there is an opportunity for an investor who can recognize these conditions to make money by buying an underpriced stock or selling an overpriced stock and perhaps buying it back later. Buying and selling options is another way of making money on the use of this information.

The fuzzy neural network used by Hobbs and Bourbakis is a modification of S. Y. Kung's fuzzy-based neural network (Kung, 1993). The inputs are 13 market indexes provided by the various financial services and institutions that reflect the average stock price. The program has consistently averaged over 20% annual return.

14.14 COMMERCIAL NEUROFUZZY SYSTEM SOFTWARE

Several software products are currently available to help with neurofuzzy problems. Four of these systems will be briefly described on the basis of information provided to the authors by the commercial organization involved. They are listed alphabetically by their commonly accepted name.

ANFIS. Jang has described ANFIS, an acronym for *adaptive neuro-fuzzy inference system*. It has an architecture that is equivalent to a two-input first-order Sugeno fuzzy model with nine rules, where each input is assumed to have three associated membership functions (Jang and Sun, 1995). Its two-dimensional input space is partitioned into nine overlapping fuzzy regions, each of which is governed by fuzzy *if-then* rules, where the premise part of a rule defines a fuzzy region, and the consequent part specifies the output within this region. ANFIS can achieve a highly nonlinear mapping. It consists of fuzzy rules which are local mappings instead of global ones. It can also be used as a neurofuzzy controller. ANFIS is implemented in the Fuzzy Systems Toolbox of MATLAB, a commercial software package produced by MathWorks, Inc.

CUBICALC. CUBICALC is a "fuzzy shell" that has great flexibility in building various kinds of fuzzy systems for decisions, inference, and control. Its primary basis for being listed here is that it has a library of neural network subroutines that can be utilized in a way that makes it possible to construct neurofuzzy systems (Watkins, 1993).

NEUFUZ. Khan (1993) reported a novel method of combining neural nets with fuzzy logic. The combined technology, NeuFuz, generates membership functions as well as fuzzy *if/then* rules by learning the system behavior using input–output data. The generated rules and membership functions are then processed using new fuzzy logic algorithms for defuzzification, rule evaluation, and antecedent processing which are developed based on neural network architecture and learning. These fuzzy logic algorithms replace conventional heuristic fuzzy logic algorithms and enable full mapping of neural net to fuzzy logic. Full mapping provides an important key feature of generating fuzzy rules and membership functions to meet a prespecified accuracy level. Simulation results have shown the approach to significantly improve performance and reliability while reducing design time and computational cost.

O'INCA. Intelligent Machines, Inc. has produced O'INCA (1994), an integrated platform for the development of fuzzy logic, neural networks, and neurofuzzy systems (O'Inca, 1993). It allows for user-defined modules in the same framework as the other systems. It combines graphical user interface, design validation, simulation and debugging, C code generation, and design documentation. In the fuzzy logic module, intermediate results after fuzzification, rulebase evaluation (inference), and defuzzification can be examined. "Fired" (active) rules can be isolated, and the results of the rule antecedent and consequent parts, rule weights, and the effects can also be examined. In the neural network module, output and bias values of all neurons, as well as all link weights, can be examined. Fixed weights and biases can be modified during simulation.

REFERENCES

Abe, S., and Lan, M. S., Function Approximator Using Fuzzy Rules Extracted Directly from Numerical Data, in *Proceedings of the International Joint Conference on Neural Networks*, Vol. 2, IEEE, Piscataway, NJ, 1993, pp. 1887–1892.

Adeli, H., and Hung S. L., *Machine Learning-Neural Networks, Genetic Algorithms and Fuzzy Systems*, John Wiley & Sons, New York, 1995.

Barbosa, V. C., Machado, R. J., dos Liporace, F., Neural System for Deforestation Monitoring on Landsat Images of the Amazon Region, *International Journal of Approximate Reasoning*, Vol. 11, No. 4, 1994, pp. 321–359.

Bezdek, J. C., and Pal, S. K. (eds.), *Fuzzy Models for Pattern Recognition*, IEEE Press, New York, 1992.

Blanco, A., Delgado, M., and Requena, I., A Learning Procedure to Identify Weighted Rules by Neural Networks, *Fuzzy Sets and Systems*, Vol. 69, 1995, pp. 29–36.

Blanco, A. and Delgado, M., Direct Fuzzy Inference Procedure by Neural Networks, *Fuzzy Sets and Systems*, Vol. 58, No. 2, pp. 133–141, 1993.

Bose, B. K., Expert System, Fuzzy Logic, and Neural Network Applications in Power Electronics and Motion Control, *Proceedings of the IEEE*, Vol. 82, No. 8 Aug pp. 1303–1323, 1994.

Brotherton, T., Pollard, T., Simpson, P., and DeMaria, A., Echocardiogram Structure and Tissue Classification Using Hierarchical Fuzzy Neural Networks, *Proceedings —ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*, Vol. 2, IEEE, Piscataway, NJ, 1994, 94CH3387-8. pp. 573–576.

Chen, C.-L, and Chen, W.-C., Fuzzy Controller Design by Using Neural Network Techniques, *IEEE Transactions on Fuzzy Systems*, Vol. 2, No. 3, pp. 235–244, 1994.

Chen, T., Lin, W.-C., and Chen, C.-T., Fuzzy Neural Networks for 3-D Heart Motion Understanding, in *Proceedings 12th International Conference on Pattern Recognition*, Vol. 2, Jerusalem, Oct. 9–13, 1994, pp. 510–512.

Chu, C.-K. P., and Mendel, P., and Jerry, M., First Break Refraction Event Picking Using Fuzzy Logic Systems, *IEEE Transactions on Fuzzy Systems*, Vol. 2, No. 4, pp. 255–266, 1994. Conference (ANNIE '94); St. Louis, MO, November 13–16, 1994.

Dalton, J. C., Adaptive Learning Systems and Qualitative Manipulation of Digital Imagery, in *Proceedings of SPIE—The International Society for Optical Engineering* Vol. 2179 Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, 1994, pp. 440–451.

El-Sharkawi, M. A., and Huang, S. J., Ancillary Techniques for Neural Network Applications, *Proceedings of the IEEE International Conference on Neural Networks*, Vol. 6, Orlando, FL., June 27–29, 1994, pp. 3724–3729.

Foslien, W. and Samad, T., Fuzzy Controller Synthesis with Neural Network Process Models, in *Proceedings of the 1993 IEEE International Symposium Intelligence Control*, 1993, pp. 370–375.

Fatikov, S., and Wohlke, G., Neuro-Fuzzy Control Approach for Intelligent Microrobots, *Proceedings of the IEEE International Conf. of Systems, Man and Cybernetics*, Vol. 4, La Touquet, France, October 17–20, 1993, pp. 441–446.

Goode, P, V., and Chow, M.-Y., Hybrid Fuzzy/Neural System Used to Extract Heuristic Knowledge from a Fault Detection Problem, in *Proceedings of the 3rd IEEE Conference on Fuzzy Systems*, Part 3 (of 3), Orlando, FL, June 26–29, 1994.

Guo, Z., and Uhrig R. E., Using Modular Neural Networks to Monitor Accident Conditions in Nuclear Power Plants, in *Proceedings of the SPIE Technical Symposium on Intelligent Information Systems*, Application of Artificial Neural Networks III, Orlando, FL, April 20–24, 1992.

Hanes, M. D., Ahalt, S. C., and Orin, D. E., Intelligent Control of Object Acquisition for Power Grasp, *IEEE International Symposium on Intelligent Control—Proceedings 1994*, pp. 303–308, 1994.

Hayashi, I., Nomura, H., Yamasaki, H. and Wakami, N., Construction of Fuzzy Inference Rules by NDF and NDFL, *International Journal of Approximate Reasoning*, Vol. 6, pp. 241–266, 1992.

Hobbs A., and Bourbakis, N. G., A NeuroFuzzy Arbitrage Simulator for Stock Investing, in *Proceedings of the IEEE/IAFE 1995 Computational Intelligence for Financial Engineering*, New York, April 9–11, 1995.

Hsiao, C.-H.; Lin, C.-T., and Cassidy, M., Application of Fuzzy Logic and Neural Networks to Automatically Detect Freeway Traffic Incidents, *Journal of Transportation Engineering* Vol. 120, No. 5, pp. 753–772, 1994.

Ichihashi, H., Miyoshi, T., and Nagasaka, K., Computed Tomography by Neurofuzzy Inversion, in *Proceedings of 1993 International Joint Conference on Neural Networks*, Part 1 (of 3), Nāgoya, Japan, Oct. 25–29 1993, pp. 709–712.

Intelligence Machines, *NeuFuz, Neural Fuzzy for Intelligence Systems*, National Semiconductor leaflet #633100, 1993.

Ishibuchi, H., Nozaki, Ken., Tanaka, H., Hosaka, Y., and Matsuda, M., Empirical Study on Learning in Fuzzy Systems by Rice Taste Analysis, *Fuzzy Sets and Systems*, Vol. 64, No. 2, pp. 129–144, 1994.

Jang, J-S. R., and Sun, C.-T., Neuro-fuzzy Modeling and Control, in *Proceedings of the IEEE*, 1995, pp. 378–406.

Keller, J. M., and Tahani, H., Implementation of Conjunctive and Disjunctive Fuzzy Logic Rules with Neural Networks, *International Journal of Approximate Reasoning*, Vol. 6, pp. 221–240, 1992.

Keller, J. M., Hayashi, Y., and Chen, Z., Additive Hybrid Networks for Fuzzy Logic, *Fuzzy Sets and Systems*, Vol. 66, No. 3, pp. 307–313, 1994.

Khan, E., NeuFuz: An Intelligent Combination of Fuzzy Logic with Neural Nets, in *Proceedings of the International Joint Conference on Neural Networks*, Vol. 3, IEEE, pp. 2945–2950, 1993.

Kim, J. S., and Cho, H. S., Fuzzy Logic and Neural Network Approach to Boundary Detection for Noisy Imagery, *Fuzzy Sets and Systems*, Vol. 65, No. 2–3, pp. 141–159, 1994.

Kim, Y. S., Mitra, S., Adaptive Integrated Fuzzy Clustering Model for Pattern Recognition, *Fuzzy Sets and Systems*, Vol. 65, No. 2–3, pp. 297–310, 1994.

Kosko, B., Fuzzy Systems as Universal Approximators, *IEEE Transactions on Computers*, Vol. 43, No. 11, pp. 1329–1333, 1994.

Kozma, R., Sato, S., Sakuma, M., and Kitamura, M., Detecting Unexperienced Events Via Analysis of Error Propagation in a Neurofuzzy Signal Processing System, *Proceedings of the Artificial Neural Networks in Engineering Conference* (ANNIE '94), St. Louis, MO, Nov 13–16, 1994.

Kulkarni, A. D., Coca, P., Giridhar, G. B., and Bhatikar, Y., Neural Network Based Fuzzy Logic Decision System, in *Proceedings of World Congress on Neural Networks*, 1994 INNS Annual Meeting, San Diego, CA, June 5–9, 1994.

Kung, S. Y., *Digital Neural Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

Kuo, R. J., Intelligent Robotic Die Polishing System Through Fuzzy Neural Networks and Multisensor Fusion, in *Proceedings of the International Joint Conference on Neural Networks*, Vol. 3, 1993, pp. 2925–2928.

Kuo, R. J., Cohen, P. H., and Kumara, S. R. T., Neural Network Driven Fuzzy Inference System, *IEEE International Conference on Neural Networks—Conference Proceedings* 3, IEEE, Piscataway, NJ, 1994, 94CH3429-8, pp. 1532–1536.

Li, W., Wu, Z., Self-Organizing Fuzzy Controller Using Neural Networks, Computers in Engineering, in *Proceedings of the International Conference and Exhibit*, Vol. 2, ASME, New York, pp. 807–812.

Lin, C.-T., and Lee, C. S. G., Reinforcement Structure/Parameter Learning for Neural-Network-Based Fuzzy Logic Control Systems, *IEEE Transactions on Fuzzy Systems*, Vol. 2, No. 1, pp. 46–63, 1994.

Lin, Y., and Cunningham, G. A., A New Approach to Fuzzy-Neural System Modeling, *IEEE Transactions on Fuzzy Systems*, Vol. 3, No. 2, pp. 190–198, 1995.

Matsuoka, K., Blanco, M. A., Neural-Fuzzy Systems for Real-Time Control of Large-Scale Utility Power Plants, in *Intelligent Control Systems American Society of Mechanical Engineers, Dynamic Systems and Control Division*, Publication DSC Vol. 48 1993, ASME, New York, 1993, pp. 75–85.

Mitra, S., Pal, S. K., Self-Organizing Neural Network as a Fuzzy Classifier, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 24, No. 3, pp. 385–399, 1994.

Moganti, M., Dagli, C., and Ercal, F., PCB Inspection Using Competitive Learning and Fuzzy Associative Memories, *Proceedings of Artificial Neural Networks in Engineering* (*ANNIE '94*), St. Louis, MO, Nov. 13–16. 1994, pp. 421–426.

Moon, S. K., Chang, S. H., Classification and Prediction of the Critical Heat Flux Using Fuzzy Theory and Artificial Neural Networks, *Nuclear Engineering and Design*, Vol. 150, No. 1, pp. 151–161, 1994.

Nie, J., Neural Approach to Fuzzy Modeling, in *Proceedings of the American Control Conference*, Vol. 2, American Automatic Control Council, Green Valley, AZ, 1994, 94CH3390-2, pp. 2139–2143.

O'INCA Design Framework, *User's Manual*, Intelligent Machines, Sunnyvale, CA, 1994, pp. 727–748.

Pao, Y.-H., Process Monitoring and Optimization for Power Systems Applications, in *Proceedings of the 1994 IEEE International Conference on Neural Networks*, Part 6 (of 7), Orlando, FL, June 27–29, 1994.

Patrikar, A., Provence, J., Control of Dynamic Systems Using Fuzzy Logic and Neural Networks, *International Journal of Intelligent Systems*, Vol. 8, No. 6, 1993, pp. 727–748.

Pham, D. T., Bayro-Corrochano, E. J., Self-Organizing Neural-Network-Based Pattern Clustering Method with Fuzzy Outputs, *Pattern Recognition*, Vol. 27, No. 8, pp. 1103–1110, 1994.

Purushothaman, G., Karayiannis, N. B, Feed-Forward Neural Architectures for Membership Estimation and Fuzzy Classification, in *Proceedings of the Artificial Neural Networks in Engineering Conference (ANNIE '94)*, St. Louis, MO, November 13–16, 1994.

Rao, D. H., Gupta, M. M., Neuro-fuzzy Controller for Control and Robotics Applications, *Engineering Applications of Artificial Intelligence*, Vol. 7, No. 5 1994, pp. 479–491.

Raza, H., Ioannou, P., and Youssef, H. M., Surface Failure Detection for an F/A-18 Aircraft Using Neural Networks and Fuzzy Logic, in *IEEE International Conference on Neural Networks—Conference Proceedings*, Vol. 5, IEEE, Piscataway, NJ, 1994, 94CH3429-8, pp. 3363–3368.

Schwartz, D. G., and Klir, G. J., Fuzzy Logic Flowers in Japan, *IEEE Spectrum*, July 1992.

Sharaf, A. M., and Lie, T. T., Neuro-fuzzy Hybrid Power System Stabilizer, *Electric Power Systems Research*, Vol. 30, No. 1, pp. 17–23, 1994.

Stylios, G. and Sotomi, O. J., Neuro-fuzzy Control System for Intelligent Sewing Machines, in *IEE Conference Publication*, No. 395, IEE, Stevenage, England, 1994, pp. 241–246.

Tascillo, A., Crisman, J., and Bourbakis, N., Intelligent Control of a Robotic Hand with Neural Nets and Fuzzy Sets, in *Proceedings IEEE Conference on Intelligent Control*, Chicago, August 1993.

Tsoukalas, L. H., Ikonomopoulos, A., and Uhrig, R. E., Virtual Measurements Using Neural Networks and Fuzzy Logic, in *Proceedings of American Power Co* [illegible] Vol. 54-II, pp. 1437–1442, 54th Annual Meeting, Chicago, April 13–15, 1992.

Uhrig, R. E., Tsoukalas, L. H., and Ikonomopoulos, A., Application of Neural Networks and Fuzzy Systems to Power Plants, in *Proceedings of the 1994 IEEE International Conference on Neural Networks*, Vol. 2, Part 6 (of 7), Orlando, FL, June 27–29, 1994, IEEE, Piscataway, NJ, 1994, 94CH3440-5, pp. 510–512.

Wakami, N., Araki, S., and Nomura, H., Recent Applications of Fuzzy Logic to Home Appliances, Plenary Session, Emerging Technologies, and Factory Automation *IECON Proceedings* (Industrial Electronics Conference), Vol. 1, IEEE, Computer Society Press, Los Alamitos, CA, 1993, 93CH3234-2, pp. 155–160.

Wang, F.-Y., Chen, D. D., in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, IEEE, Piscataway, NJ, 1994, 94CH3571-5, pp. 1803–1808.

Wang, L.-X., *Adaptive Fuzzy Systems and Control*, Prentice Hall, Englewood Cliffs, NJ, 1994.

Watkins, F., "Tutorial, Neural fuzzy Systems" notes, *World Congress on Neural Networks, 1993 International Neural Network Society Annual Meeting*, Portland OR, July 1993.

Wegmann, H., Fuzzy Control and Neural Networks Industrial Applications in the World of PLCs, in *Proceedings of the IEEE Conference on Control Applications*, Vol. 2, IEEE, Piscataway, NJ, 1994, 94CH3420-7, pp. 1245–1249.

Werbos, P. J., Neurocontrol and Fuzzy Logic: Connections and Designs, *International Journal of Approximate Reasoning*, Vol. 6, pp. 185–219, 1992.

Werbos, P. J. Elastic Fuzzy Logic: A Better Way to Combine Neural and Fuzzy Capabilities, in *Proceedings of the IJCNN*, Vol. II, Portland, OR, pp. 623–626, 1993.

Yager, R. R., Modeling and Formulating Fuzzy Knowledge Bases Using Neural Networks, *Neural Networks*, Vol. 7, No. 8, pp. 1273–1283, 1994.

Yamazaki, K., Kaneko, H., Yamaguchi, S., Watanabe, K. Y., Taniguchi, Y., and Motojima, O., Design of the Central Control System for the Large Helical Device (LHD), in *Nuclear Instruments and Methods in Physics Research*, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Vol. 352, No. 1–2, December 15, 1994, pp. 43–46.

Yen, G. G., Hybrid Learning Control in Flexible Space Structures with Reconfiguration Capability, in *IEEE International Symposium on Intelligent Control—Proceedings 1994*, IEEE, Piscataway, NJ, 1994, 94CH3453-8, pp. 321–326.

15

DYNAMIC HYBRID NEUROFUZZY SYSTEMS

15.1 INTRODUCTION

In the decade of the 1990s, the authors and their graduate students[1] have carried out research activities that utilize hybrid systems involving various combinations of neural network, fuzzy systems, genetic algorithms, and expert systems. These hybrid systems were utilized to acquire and process data from engineering systems, ranging from nuclear and fossil power plants to steel rolling mills, as well as their various components (check valves, compressors, rolling element bearings, control systems, etc.) The purpose of this chapter is not to describe the result of this work, but rather to convey the essence of the methodologies developed and how they were used advantageously in comparison with more conventional technologies. Hence, only those details necessary to illustrate the use of hybrid artificial intelligence techniques are presented. Although the results of the work are not given, they can be obtained from the references cited.

Various aspects of the analysis of data from three specific sets of experiments are involved in the work described here. The first is a set of vibration spectral data provided by Électricité de France on accelerated testing of bearings with faults deliberately introduced in some bearings to induce early

[1] Special acknowledgment must be given to Dr. Andreas Ikonomopoulos, on leave from the Demokritos Nuclear Research Laboratory, Athens, Greece; Dr. Anna Loskiewicz-Buczak, now with AlliedSignal Corporation, Morristown, NJ, and Dr. Israel E. Alguindigue of the University of Tennessee, Chattanooga, who worked with the authors on many of the activities described in this chapter while pursuing their doctoral degrees at the University of Tennessee, Knoxville, TN.

failure.[2] The diagnosis of faults in roller bearings is based on the relative magnitude of the peaks occurring in spectra at characteristic frequencies (and their harmonics) associated with the bearing geometry and the basic rotating frequency. A comparison between the results using crisp magnitudes of the amplitudes and two types of fuzzy representation of these amplitudes is presented. The goal was to detect which bearings had faults and to identify the magnitude and location (inner race, outer race, or ball) of the faults from the vibration spectra measured by accelerometers mounted on the frame supporting the bearing (Loskiewicz-Buczak, 1993).

The second is a set of vibration spectral data[3] taken from "laminar flow" table rolling machines in a steel sheet manufacturing mill. Data were taken from sensors at nine locations on each of 163 table rolling machines, 49 of which had one or more faults identified. In this example, a composite diagnosis of single and multiple faults in the machines is obtained based on the fusion of nine tentative diagnoses (which were usually all not the same) indicated by nine neural networks processing data from nine sensors placed on the individual machine. The fusion of these decisions is performed by a fuzzy logic connective called the generalized mean (a generalized type of fuzzy variable). The goal of the study was to fuse the data together using neural networks and fuzzy systems methodologies to identify the faults (Loskiewicz-Buczak and Uhrig, 1994).

The third is a set of data taken from the High Flux Isotopes Reactor (HFIR) at Oak Ridge National Laboratory[4] during startup from source level to full power. The goal here is to demonstrate the interrelationship between the various output variables at the HFIR and to infer the value of a variable that cannot be measured directly (Tsoukalas, 1993; Ikonomopoulos et al., 1994).

The final section deals with various aspects of neurofuzzy control, including some discussion of neurofuzzy approaches to anticipatory control. Fuzzy logic and neural networks are complementary technologies, and both are well suited for controlling nonlinear and time-varying system. This discussion reviews the benefits of integrating these two methodologies in an advantageous way. Anticipatory control is one of the areas that can benefit from a neurofuzzy approach. The ability to predict the future faster than real time

[2] The authors are indebted to Électricité de France for providing the data used here from tests carried out at their laboratory facilities near Paris. This work was performed as part of a contract with Électricité de France carried out by one of the authors and his graduate students at the University of Tennessee.

[3] The authors are indebted to the U.S. Steel Corporation, Gary, IN, for permission to utilize vibration spectra data from rolling mills in their plants and to Technology for Energy Corporation, Knoxville, TN, for gathering these data and making them available to the authors and their graduate students.

[4] The authors want to acknowledge the efforts made by many individuals at Oak Ridge National Laboratory to make this project possible.

enables us to take steps to correct a deteriorating situation (Tsoukalas et al., 1994a,b).

15.2 FUZZY-NEURAL DIAGNOSIS FOR VIBRATION MONITORING

Vibration monitoring of engineering systems involves the collection of vibration data from system components and detailed analysis to detect features which reflect the operational state of the machinery. The analysis leads to the identification of potential failures and their causes and makes it possible to perform timely preventive maintenance. A hybrid neurofuzzy system for vibration monitoring of rolling element bearings is discussed. The system takes advantage of the learning and generalization abilities of neural networks. The ambiguity that accompanies fault diagnosis is handled by means of fuzzy membership functions. The combination of neural networks and fuzzy logic contributes to the high speed and flexibility of the system.

For many machines, the vibration frequency spectrum has a characteristic shape when the machine is operating properly, and it has other features for different faults that may appear. Recognition of faults can be accomplished in many cases by detecting specific features in the frequency spectrum which are known to be related to particular faults. All vibration monitoring techniques are based fundamentally on the recording and quantification of small vibration impulses (Zwingelstein and Hamon, 1990). Often, spectral features associated with specific defects are generated at frequencies that can be calculated from formulae derived from bearing geometry.

However, the task of recognition is complicated by a series of factors, such as noise, presence of multiple faults,'severity of the fault and speed changes. Fault recognition is also complicated by the fact that fundamental frequency components often disappear at advanced stages of the defect, while harmonic components remain. Furthermore, when performing vibration monitoring of rolling element bearings, the emphasis is more on the content of the spectrum than on its amplitude (Hewlett Packard, 1983; Berry, 1990) (Jackson 1979). Amplitudes of bearing characteristic frequencies often begin to decrease as conditions worsen. Therefore, more importance should be attributed to the fact that a multiple number of fault frequencies are appearing in the spectrum than to the exact amplitude. This fact led to incorporation of fuzzy logic into the classification system. The soft boundaries in fuzzy logic environments, obtained by membership functions, are of special interest because their use results in flexible, more human-like classifications. On the other hand, neural networks provide a viable technique for the analysis of vibration data because of their inherent ability to operate on noisy, incomplete, or sparse data and to model processes from actual system parameters. Some previous University of Tennessee work (Loskiewicz-Buczak and Uhrig, 1992, 1993a, b; Alguindigue et al., 1993) deals with the problem of vibration monitoring by neural network technology alone. By combining neural networks and fuzzy logic, we are able to take advantage of the strengths of both approaches (Loskiewicz-Buczak, 1993a, b).

Vibration Signatures

To perform spectral monitoring of components in an operating engineering system, signatures are collected from plant machinery and analyzed to detect features which reflect the operational state of the machinery. The data consist of vibration measurements collected from SKF ball bearings of type 6206 during an aging simulation process. The rolling element under test is mounted on a horizontal shaft and loaded radially by means of a jack, imposing a vertical force on the bearing. These severe conditions generate scaling faults on the component. Data are collected using an accelerometer placed in the radial direction to the loading zone of the bearing. From these measurements, spectra are generated using fast Fourier technique transform (FFT) techniques (see Figure 15.1). Spectra are averaged over 16 samples with a Hanning window, and each contains 397 points in the range 5 Hz to 1 kHz.

Methodology

For this project, first the characteristic frequencies for a flaw in the inner and outer races and in one of the balls can be calculated in terms of its rotational

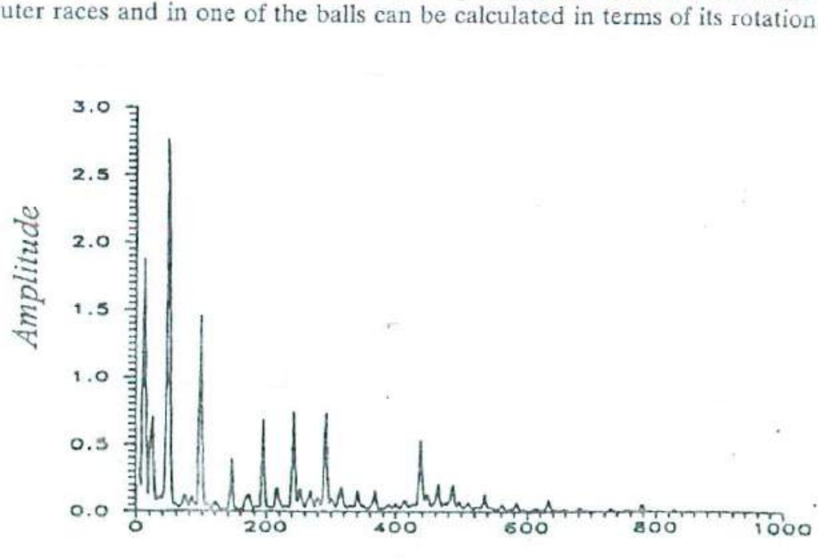


Figure 15.1 Spectrum of a healthy 6206 bearing.

speed from its dimensions (Berry, 1990). These critical frequencies for the type 6206 bearing manufactured by SKF were calculated. Then the location of the peaks at these characteristic frequencies as well as at their harmonics was investigated. The exact value of the amplitude of each peak is not important for the classification process. Instead, we need to know if there is a peak at a given frequency and whether it is small or big. Therefore a transformation of peaks' amplitudes by fuzzy membership functions was performed. At the beginning three triangular membership functions—"none," "small," and "big" (see Figure 15.2)—were used. Then four membership functions were tried using "non," "small," "medium," and "big" (See Figure 15.3). These values of membership functions are the input to the Kohonen self-organizing map with categorization. The output of the network is the fault (or faults) present. The effect of different number of membership functions on the final classification was investigated.

For the analysis of spectral signatures, neural networks may be used both as classifying and clustering systems. To perform classification it is necessary to attach to each signature a label which describes the operational state of the machine at the time of collecting the signature. The input to the network is a spectrum, or some features from it, and the output is the class label. The network is trained to identify an arbitrary pattern as a member of a state among a set of possible states. Clustering involves the grouping of patterns according to their internal similarity and is performed in an unsupervised mode. The aim of clustering is to distribute the set of patterns into states such that the patterns in each state have similar statistical and geometrical properties.

For this project a two-dimensional self-organizing map (SOM) neural network was used (Kohonen, 1990; NeuralWare, 1991). In order for the SOM

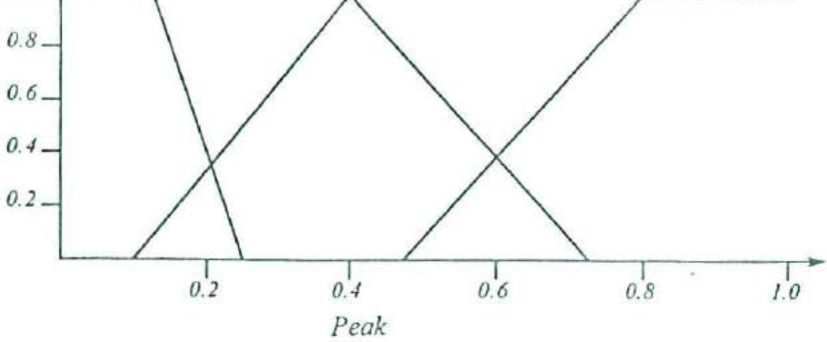


Figure 15.2 Membership functions for "*none*," "*small*," and "*big*."

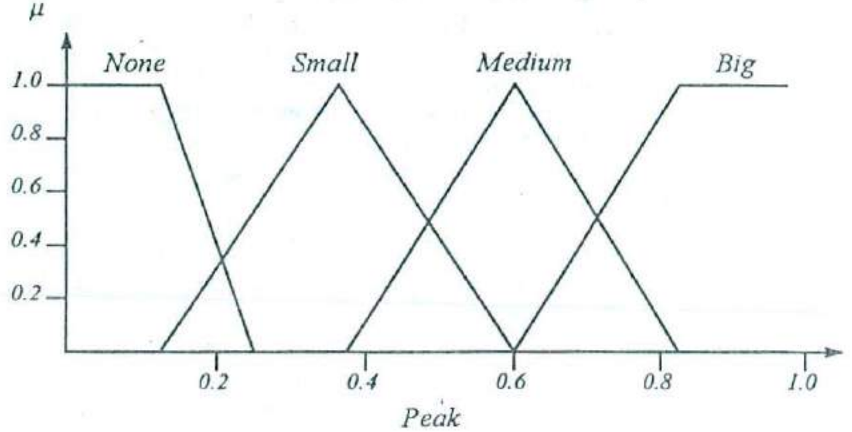


Figure 15.3 Membership functions for "*none*," "*small*," "*medium*," and "*big*."

network to solve categorization problems, an output layer is added to the network. During training the SOM is given a sufficient number of iterations in which to stabilize with the learning rate of the weights going to the output layer set to zero. Hence, the network begins training in an unsupervised mode and then uses supervised training for the output layer.

Results

For this project the NeuralWorks™ version of SOM with a categorization network (NeuralWare, 1991) was used. Three different kinds of inputs to the networks were compared. As the first type of input, amplitudes transformed by three membership functions (see Figure 15.2) were used. The second type of input consisted of amplitudes transformed by four membership functions. (see Figure 15.3.). The last type of input were "raw" amplitudes (without the membership function transformation). In each case, 21 signatures (29.6% of the whole set) were used for training, and the entire set (71 signatures) was used for recall. The output layer of the network was the same in all the situations: six output nodes, each one corresponding to one fault. An activation of 0.5 in an output node indicated the presence of the corresponding fault. Activation in more than one node corresponded to a multiple fault, while no activation was perceived as no fault.

For the transformation of amplitudes by three membership functions the network has 48 inputs $(16 * 3)$. This corresponds to three membership values for each of 16 amplitudes at frequencies related to faults. Different sizes of the Kohonen map were tried: 4×5, 5×5, 6×5, 6×6, 6×7. The best results were obtained with the 6×5 Kohonen layer. In this case, only one misclassification occurred (98.5% accuracy). One of the signatures used in training was classified as nonfaulty, whereas it was an instance of a generalized scaling fault of all the components. Had only the signatures not used in training been classified, the accuracy would have been 100%.

When using four membership functions for the transformation process, the input layer had 64 (16 * 4) nodes. The best results were obtained with the Kohonen layer 6 × 5, resulting in misclassification of one signature. This signature, instead of being classified as a localized fault on the outer race, was classified as nonfaulty. The results for three and four membership function transformations give the same accuracy. However, for four membership functions many of the nonfaulty signatures have one of the outputs with value about 0.3, whereas when three membership functions are used the outputs for nonfaulty signatures are at most 0.05. This means that the transformation by three membership functions gives more robust results than the transformation by four functions.

When using the raw amplitudes as the input to the neural network, the input layer has 16 nodes. The best results were obtained also for the 6 × 5 Kohonen layer. However, even in this case, 12 of the 71 signatures were misclassified, which gives only 83.1% classification accuracy. The transformations of both fuzzy membership functions results in great improvements of the final classification over a neural network that uses raw amplitudes as inputs.

Fuzzy c-Means Clustering Algorithm

The fuzzy c-means algorithm has been used for the clustering and classification of vibration signatures in the frequency domain (Alguindigue et al., 1992). The fuzzy c-means algorithm is a variant of the fuzzy clusteing algorithms pioneered by Bezdeck since the late 1970s. It attempts to cluster measurement vectors by searching for local minima of the generalized within group sum of squared errors functions (WGSSE). It was proposed by Trivedi and Bezdeck (1986) and is given by

$$J_m(\mathrm{U},\mathbf{v}) = \sum_{k=1}^{n} \sum_{i=1}^{c} (u_{ik})^m |x_k - v_i|_A^2, \qquad 1 < m < \infty \qquad (15.2\text{-}1)$$

where c is the number of clusters, n is the number of vectors, x_k is a kth measurement vector, v_i is the ith centroid vector, m is the fuzzy coefficient, $|\cdot|_A$ is an inner product norm, $|Q|_A^2 = Q^T A Q$, and A is a $d \times d$ positive definite matrix where d is the dimension of the pattern vectors.

When $m = 1$ the objective function J_m in (15.2-1) is the classical WGSSE function, and the algorithm reduces to the crisp k-means clustering algorithm. For $m > 1$ under the assumption that $x_k \neq v_i$, $(\mathrm{U},\mathbf{v})$ may be a local minimum of J_m only if

$$u_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{\|x_k - v_i\|_A}{\|x_k - v_j\|_A} \right)^{2/(m-1)}} \qquad \forall\, i, k \qquad (15.2\text{-}2)$$

and

$$v_j = \frac{\sum_{k=1}^{n} (u_{ik})^m x_k}{\sum_{k=1}^{n} (u_{ik})^m}, \qquad \forall\, i \qquad (15.2\text{-}3)$$

The fuzzy c-means algorithm consists of the following steps (Trivedi and Bezdeck, 1986):

1. Fix the number of clusters c. Select the inner product norm. Fix the fuzzy coefficient m. Set $p = 1$ and initialize $U^{(0)}$.
2. Calculate fuzzy cluster centers $\{\mathbf{v}^{(p)}\}$ using $U^{(p-1)}$ and the condition specified in equation (15.2-3).
3. Update $U^{(p)}$ using $\mathbf{v}^{(p)}$and the condition specified in equation (15.2-2).
4. If $\|U^{(p)} - U^{(p-1)}\|_A < \varepsilon$ then terminate; else set $p = p + 1$ and go to step 2.

15.3 DECISION FUSION BY FUZZY SET OPERATIONS

Fusion of information from multiple sources for object recognition and classification is an increasingly important area of research and application. Information fusion is employed in robotics, computer vision, managerial decision-making systems, and many engineering systems. Fusion of information is often made more difficult by problems of uncertainty characterized by vagueness, inexactness, and ill-definedness. This is the reason to employ fuzzy set theory in information fusion systems.

Vibration Signatures

Data used for this project consist of vibration signatures from 163 identical "laminar flow" table rolls in a steel sheet manufacturing mill. Data were collected with sensors attached to the plant machinery at the same nine locations on each machine. Spectra acquired from the nine sensors are correlated but not identical due to different vibration levels throughout the machine and to the fact that the faults which are particular to a bearing located near one sensor are not necessarily recorded by the other more distant sensors. The 150-point spectrum of each sensor output is generated using FFT techniques, and the coefficients are stored in a database. The data set contains signatures from 49 machines for which the types of faults had been identified. For some machines, one to three sensor readings were missing. The data sets reflected faulty operating conditions such as misalignment (M), looseness (L), wear (W), outboard bearing damage (O), lubrication (C), and their combinations (double and triple faults.) Data from machines operating properly were not included in the data set used here.

The first step is classification of signatures coming from each sensor separately using recirculation neural networks to reduce dimensionality, and backpropagation or probabilistic neural networks for classification of faults. This classification process has been adequately described in the literature (Alguindigue et al., 1993; Loskiewicz-Buczak and Uhrig, 1993a, b, 1992) and will not be repeated here. The second step is information fusion from these nine classifications, performed by means of fuzzy set operations. Information fusion is used whenever several sensors are employed in a system, in order to reduce uncertainty and resolve the ambiguity often present in classifications from several sensors. In this approach, a confidence factor of the fused decision is determined and the data are fused only from the sensors which cause the confidence factor to grow.

Information Fusion by Means of Fuzzy Logic

Among the approaches for information fusion that have been proposed in the literature are probability theory, Dempster–Shafer theory, neural networks theory and fuzzy set theory. Fuzzy set theory provides several advantages due to the fact that there are numerous ways of combining fuzzy sets in addition to the union (e.g., the "max" operator) and intersection (e.g., the "min" operator) used in traditional theories. Numerous fuzzy set connectives can be used for the purpose of aggregation (Krishnapuram and Lee, 1992; Zimmermann, 1987). The requisites of the decision-making process and the character as well as relative importance of criteria determine the particular connective to be chosen. The requisite may be that all the criteria be satisfied for which an intersection connective should be used, or any one of the criteria be satisfied for which a union connective should be used. When the criteria are mutually compensatory, a mean operator is the most appropriate. Usually in decision-making based on several criteria, a certain amount of compensation is desirable. Zimmermann (1987) showed that human decisions and evaluations almost always show some degree of compensation and that the "generalized mean" used here very closely matches the human-decision making process. In almost all categorization problems the final classification that the system should give is the one that humans would give. This is the reason to use the aggregation connectives that match the best the human decision-making process for fusion of evidence.

For this project the "generalized mean" operator was chosen for the fusion process. It was proposed first by Djumovic (1974) and later Dyckhoff and Pedrycz (1984) and defined by

$$g(x_1, x_2, \ldots, x_n; p, w_1, w_2, \ldots, w_n) = \left(\sum_{i-1}^{n} w_i x_i^p\right)^{1/p} \qquad (15.3\text{-}1)$$

where p is the degree of fuzziness, and the w_i's can be thought as the relative importance factors for the different criteria where

$$w_1 + w_2 + \cdots + w_n = 1 \qquad (15.3\text{-}2)$$

The behavior of the generalized mean with p is shown in Figure 15.4 where the amplitude has been scaled between 0.1 and 0.9. The attractive properties of the generalized mean are as follows:

- $\min(a, b) \leq \text{mean}(a, b) \leq \max(a, b)$;
- mean increases with an increase in p; by varying the value of p between $-\infty$ and $+\infty$, one can obtain all values between min (intersection) and max (union) respectively.

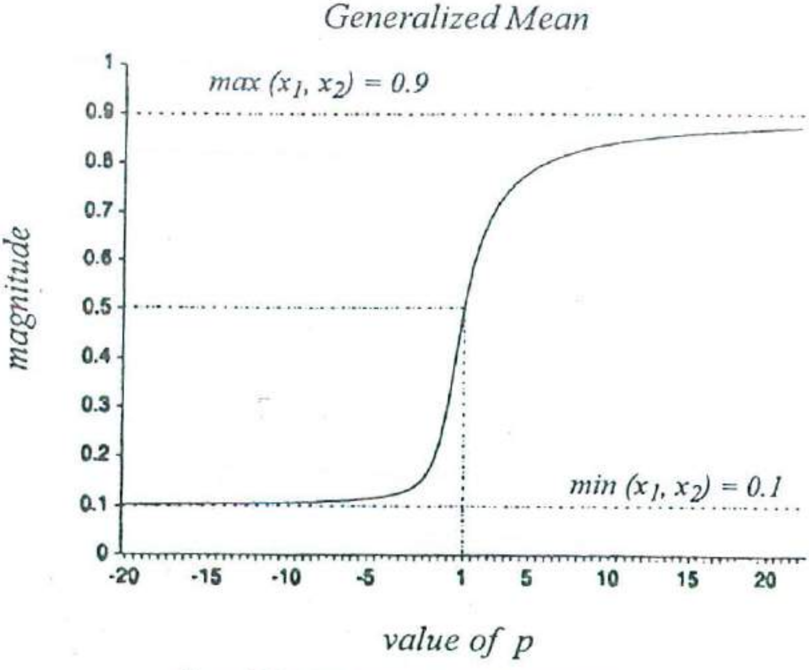


Figure 15.4 The generalized mean operator.

Therefore in the extreme cases the generalized mean operator can be used as union or intersection. Also, it can be shown that $p = -1$ gives the harmonic mean, $p = 0$ gives the geometric mean, and $p = 1$ gives the arithmetic mean. The rate of compensation for the generalized mean can be controlled by changing p. When using larger values of p, the partition becomes more fuzzy.

The definition of the confidence factor can affect the fusion results significantly. For classification problems, the confidence factor (CF) is defined as:

$$\mathrm{CF} = \frac{1}{\text{average error}} \tag{15.3-3}$$

$$\text{average error} = w_1 \cdot \text{error}^1 + w_2 \cdot \text{error}^2 \tag{15.3-4}$$

$$\text{error}^k = \sqrt{\sum_{i-1}^{n} \sum_{j-1}^{c} \left(\text{mean}_{ij} - \text{pattern}_{ij}^k\right)^2} \tag{15.3-5}$$

where n is the number of patterns, c indicates the number of classes of faults, and $k = 1, 2$ is the sensor number. The average error [equation (15.3-4)] is a measure of how different the aggregated decision is from the earlier decisions that were the input to the aggregation process. This error will become small when enough decisions are aggregated, because the decision from the next sensor will tend to be redundant with the decisions already fused.

The weights used in the calculation of the average error are the weights used for the fusion process. These weights should describe the relative confidence that we have in those sensor measurements. If there is no reason to bias the decision, all the weights should be the same. For this project, there is no information on the precision of the sensor readings, and therefore the weights for the fusion process for each sensor are the same when fusing information from the first two sensors. At later fusion steps involving n decisions, when one decision is already an aggregated decision from $n - 1$ sensors and the other is a decision from only one sensor, the weights are calculated as $(n - 1)/n$ and $1/n$, respectively. This ensures that the decision from each sensor is given the same importance.

"Active" Information Fusion Scheme

For the fusion process, one can choose a larger p value for fusion of information from complementary sensors and choose a smaller p value for fusion of information from redundant sensors. In this case, information on the degree of complementarity/redundancy of the sensors is required. However, if no such information is available (but the number of sensors is large), it is reasonable to presume that at the beginning of the fusion process [i.e., when fusing information from a small number of sensors (1, 2, 3...)], this information is complementary; as the number of sensors increases, the information should become more and more redundant. Therefore using large p (union-like operation) at the beginning of the fusion process, and decreasing p as the number of sensor increases, seems to be the most appropriate method. In the project this method was used.

Our goal is the best classification possible. We want the fusion process to be "active," meaning that the next step of the fusion process is determined by the results of the previous one. The fusion scheme is the following:

1. Fuse the decisions from two sensors.
2. Evaluate the confidence factor of the fused decision.
3. Fuse the decision from the next sensor.
4. If the confidence factor has decreased undo step 3 (do not fuse data from this sensor.)
5. If there are data from more sensors, repeat steps 3–5; otherwise, this is the aggregated decision.

As the number of sensors involved increased, the value of p in equation (15.3-1) changed from a large value for complementary sensors to a smaller value when the addition sensor information was considered redundant. Subsequent work involved the use of genetic algorithms to optimize the sequence in which the sensor decisions are fused. The concern here is that the choice of a bad sensor decision for the first fusion step could lead to the rejection of good sensor decisions in later fusion steps. Each advancement in methodology improved the resultant identification of the faults.

The final decision has to be obtained from the aggregated decision by some defuzzification method. The method chosen was α-cuts. After fixing the value of α, an α-cut is performed on the aggregated decision. For each of the five faults (M, L, W, O, C) there is a corresponding α-level set (M_α, L_α, W_α, O_α, C_α). Each of these sets includes all the patterns that are manifesting a given fault. If a pattern belongs only to one α-level set, it means that the final decision is that it is exhibiting only this fault (single fault pattern). If a given pattern belongs to more than one α-level set, it means that the final decision is that it is a multiple fault pattern, manifesting the faults to which α-level sets the pattern belongs.

15.4 HYBRID NEUROFUZZY METHODOLOGY FOR VIRTUAL MEASUREMENTS

A method of generating fuzzy numbers representing the values of system-specific variables such as performance has been developed (Ikonomopoulos et al., 1992, 1994). It constitutes essentially the fuzzification (symbolization) of measurements (and predictions), and thus we refer to it as *virtual measurement*. It should be remembered, however, that virtual measurements are simply predictions involving fuzzy numbers where the notion of a measuring device has been extended to incorporate significant modeling capability at the level of the instrument.

In virtual measurements, neural networks are used to perform a mapping

$$f: M \rightarrow E \tag{15.4-1}$$

where the domain M is the hyperspace of accessible variables such as temperatures and pressures in an engineering system, and the output range E is a set of fuzzy numbers that constitute our predictions of fuzzy values referred to as *virtual measurement values* (VMVs). (VTMs are the fuzzy analogs of the units of measure, e.g., volts, pounds, degrees, etc.) As discussed in Chapter 4, a fuzzy number is a normal and convex fuzzy set on the real numbers which models the value of a fuzzy variable at any given time, uniquely represented by a membership function. The fuzzy numbers used here had a trapezoidal shape. Trapezoidal membership functions are uniquely described by a set of four numbers—for example, a given number $C = \{o_1, o_2, o_3, o_4\}$, where $O \leq o_1, o_2, o_3, o_4 \leq 1$ and $\{o_1, o_2, o_3, o_4\}$ (from left to right) represents the universe of discourse components of the four corners of the trapezoid (from left to right). Such representations offer considerable advantage to computing speed.

The methodology for predicting fuzzy numbers used here has been described elsewhere (Ikonomopoulos et al., 1994), and its main points may be summarized in the following steps:

1. Decide how many fuzzy values are necessary to adequately cover the range of the fuzzy variable to be predicted.
2. Determine the number and the type of physically measurable variables that will be the basis (i.e., the input) of the virtual instrument.
3. Train one neural network per VMV, for example, a program trained on five VMVs will require five trained neural networks as shown in Figure 15.5.
4. Design an appropriate logic using the *index of dissemblance* to select which membership function will be the predicted value of the instrument at any given time.

The networks $N_1, N_2, \ldots, N_n$ comprising the virtual instrument are trained (in a process analogous to "calibration") with time series as input vectors, and vectors $\{o_1, o_2, o_3, o_4\}$ representing fuzzy numbers are trained as outputs. Each network learns to map a constellation of input patterns to a particular linguistic label. The situation is illustrated in Figure 15.5, where five inputs to each of the n networks are used; hence this virtual instrument is calibrated

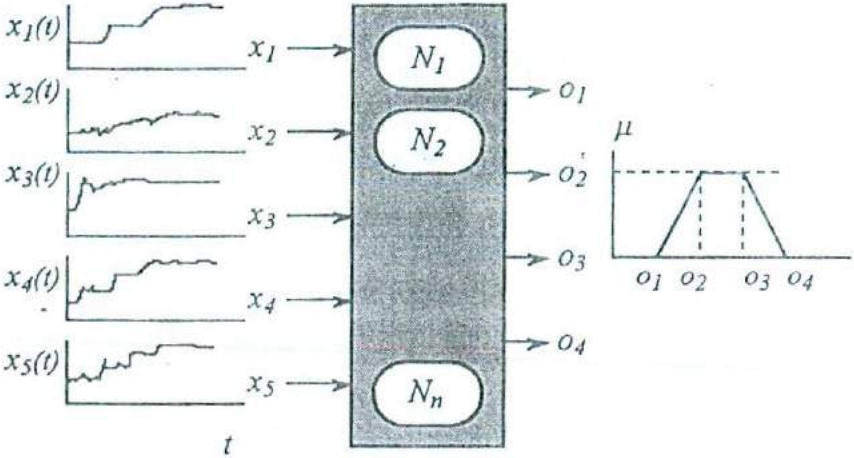


Figure 15.5 Each neural network in a virtual instrument maps a time-series input vector onto a vector $\{o_2, o_2, o_3, o_4\}$ representing a trapezoidal fuzzy number.

with n fuzzy numbers. After training, all networks $N_1, N_2, \ldots, N_n$ receive on-line time signals as inputs and produce a set of membership functions as outputs. Generally the outputs will be somewhat different from the membership functions the networks were trained for (the prototypes); moreover, one or at most two (if we allow overlap of membership functions) will represent correct values while the rest need to be ignored. It is thus important to identify the correct output. Since we consider each network's output to be a fuzzy number, we use a *dissemblance index* (Kaufmann and Gupta, 1991) to estimate the output membership functions that are closest to the set of prototype membership functions on which we trained the networks and select one as the predicted fuzzy value. The dissemblance index, $\delta(A, B)$, of two fuzzy numbers A, B gives the distance between the fuzzy numbers. When $\delta(A, B) = 0$ we can infer that the fuzzy numbers are identical. When $\delta(A, B) = 1$ we infer that the fuzzy numbers are totally different.

Using physically observable quantities to predict fuzzy values offers some unique advantages. A set of complicated time series is mapped to the universe of discourse of human linguistics through a neural network which acts as an interpreter of vital information supplied from the system. The information encoded in a time series is in the form of rate of increase/decrease and maximum/minimum values attained over a period of time. The network is trained to represent this kind of "hidden" information in the form of membership functions which can be used for fuzzy inferencing as shown by Sugeno and Yasukawa (1993). The membership function provides sufficient information to predict the value of a fuzzy variable in the near future.

Furthermore, a network trained to recognize a specific complicated time pattern (i.e.,the time series has "crisp" values) will lose much of its ability to deal with noisy input signals since it will tend, for distorted inputs, to produce averaged forms of the desired output, missing therefore vital pieces of information.

As an example of the prediction method, consider the following experiment. Actual data obtained during a start-up of the high flux isotope reactor (HFIR) was used in order to test the methodology for predicting fuzzy values. HFIR is a three-loop pressurized water research reactor operated at the Oak Ridge National Laboratory. A flow control valve on the secondary side of the system is used as the main mechanism for control (there is also a "trim flow control valve" for finer flow adjustments, as well as control rods) as shown in Figure 15.6. Although the signal sent to the motor of the valve is known, the actual position of the secondary flow control valve is not known and is rather hard to predict. The disk position is something that the operators of the plant "learn" how to estimate intuitively on the basis of experience. However, valve aging and varying plant operating conditions as well as operator experience are major factors for substantial variations in the estimate of valve position.

Five parameters in the form of time series were chosen as the basis for predicting the secondary flow control valve position: *neutron flux*, *primary flow pressure variation* (*DP*), *core inlet temperature*, *core outlet temperature*, and *secondary flow*. All but the last one of the above-mentioned time series contain average values of the corresponding parameters of the three-loop system. Figure 15.7 shows the secondary flow signal normalized in the range between zero and one. These five parameters were selected in order to provide sufficient description of conditions in both the primary and secondary sides of HFIR during start-up. The time series of these five parameters are used to train five neural networks (i.e., $n = 5$, but it can be any number dependent upon the virtual measurement values (VMV). Each one

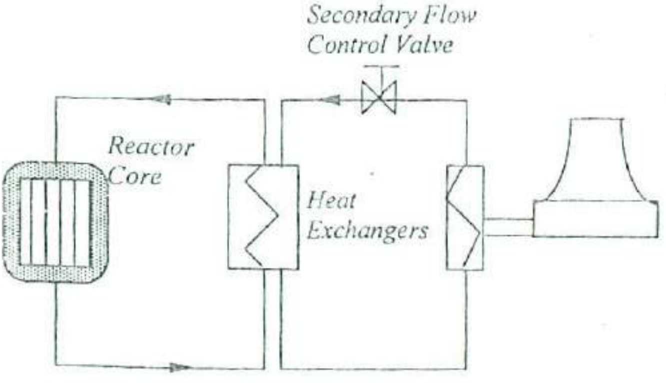


Figure 15.6 Schematic of the high-flux isotope reactor.

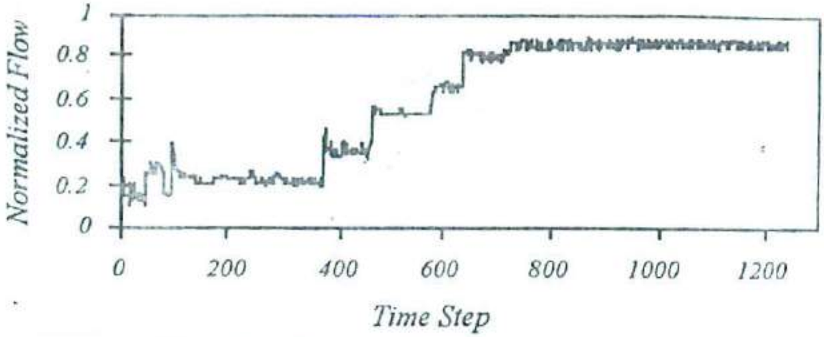


Figure 15.7 Secondary flow signal during start-up.

of them has five neurons at the input layer and four neurons at the output layer and 10 neurons in the hidden layer of each network. In each network N in Figure 15.5, there are five input neurons each receiving a time series from the five physically measurable variables; four output neurons represent the four corners of a trapezoidal membership function. The output is a membership function uniquely labeling a fuzzy value of the fuzzy variable describing the position of the secondary flow control valve, referred to as *valve_position*. The data for each of the time series used for network training is scaled to the interval 0.1 to 0.9 and sampled every 16 seconds, with a total of 1240 samples available.

Designing a virtual instrument to predict *valve_position* requires first the partition of its membership of discourse with the appropriate number of VMVs. In this example, five values—*CLOSED, PARTIALLY_CLOSED, MEDIUM and PARTIALLY_OPEN* and *OPEN*—were chosen. (The choice of five VMVs had nothing to do with the fact that there are five input variables.) Each value is represented by a membership function, namely, μ_{CLOSED}, $\mu_{PARTIALLY\_CLOSED}$, μ_{MEDIUM}, $\mu_{PARTIALLY\_OPEN}$, and μ_{OPEN}. These five membership functions describe the position of the valve at every instant during the start-up period. The universe of discourse on which these membership functions are defined is the interval [0, 1]. Thus, μ_{OPEN} associates each point in the universe of discourse with the fuzzy value *OPEN* at this point.

The membership functions representing the output of the predictive instrument in this particular study have trapezoidal shape or the degenerated (triangular) form of it, which is very useful for computations in the fuzzy control area. The membership function for *CLOSED*, namely μ_{CLOSED} is defined by a trapezoid with peak coordinates {(0.02, 0), (0.05, 1), (0.10, 1), (0.2, 0)}. Similarly, *PARTIALLY_CLOSED* is represented by the trapezoid with coordinates {(0.15, 0), (0.2, 0), (0.30, 1), (0.4, 0)}, *MEDIUM* by {(0.35, 0), (0.4, 1), (0.50, 1), (0.6, 0)}, *PARTIALLY_OPEN* by {(0.5, 0), (0.6, 1), (0.7, 1), (0.75, 0)}, and *OPEN* by {(0.7, 0), (0.82, 1), (0.85, 1), (0.90, 0)}. It is evident from the above geometrical schemes that there is an overlap between the membership functions used. The reason for the overlap is the fuzziness in the definition of the different states of valve position.

Figure 15.8 shows the prediction of the instrument during a start-up of the reactor (1240 time steps). The valve is initially *CLOSED* as seen by the membership function in the origin of the 3-D graph. It goes through the "medium" range rather quickly in the vicinity of 400–500 time steps, and finally it becomes fully open after the 800th time step. Note that this confirms rather well the trend shown in Figure 15.7 where the secondary flow reaches its maximum value after about the 800th time step.

To test the ability of each network to predict the valve position by calculating the right membership function at any particular time step, different levels of noise were introduced in the input signals. Initially up to 10% noise was introduced to all five input signals, and the set of networks was tested with the "noisy" vectors. The appropriate networks fired at the corresponding time steps, calculating the coordinates of the peaks of the corresponding membership functions with 98% accuracy. Henceforth there was an excellent prediction of the position of the disk valve during the whole time interval under consideration. In addition, 20% noise was introduced to

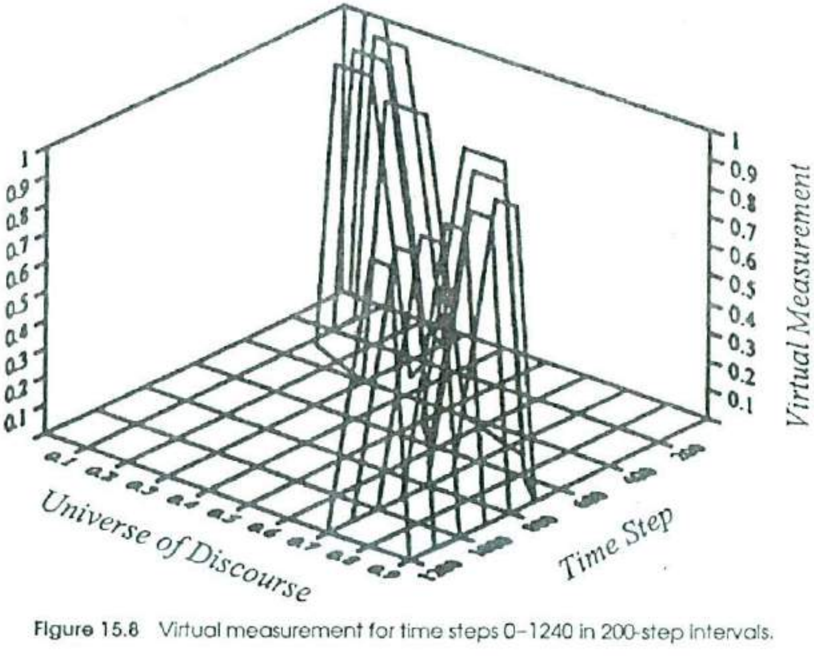


Figure 15.8 Virtual measurement for time steps 0–1240 in 200-step intervals.

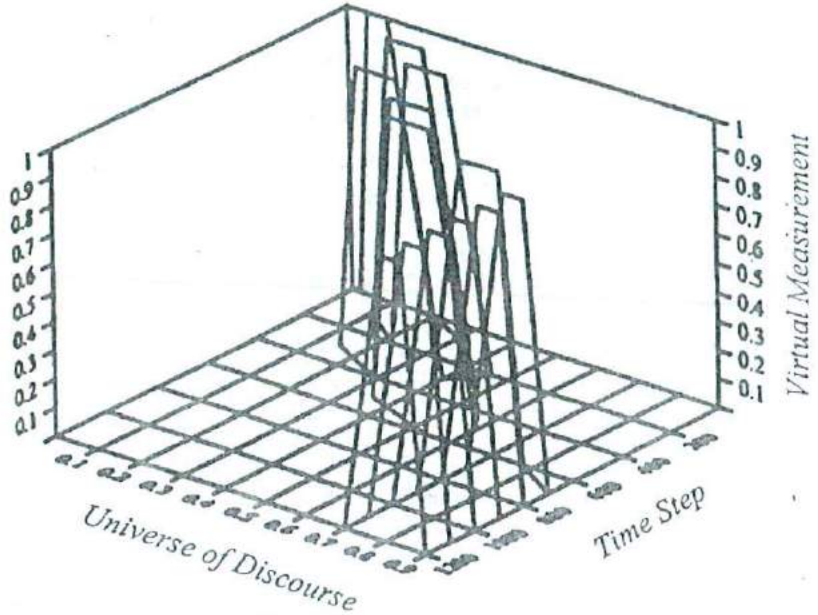


Figure 15.9 Virtual measurement for time steps 0–1240 in 200-step intervals when the secondary flow input signal has been substituted with 100% noise.

all five input signals, and the networks were tested again. The response of the system was indistinguishable form the previous case.

Even when the most closely related input signal was replaced with random noise, the predictive instrument still predicted the valve position rather accurately. Figure 15.9 shows the output of the instrument when the secondary flow signal has been replaced with random noise. Comparison with Figure 15.8 shows that the virtual instrument still indicates the valve position rather well. A series of statistical tests were conducted to confirm that the output of the instrument is actually within random error of the previous case. This represents a significant tolerance to informational hazards to which the instrument was exposed. Even with about 20% of its input information lost, it still rather accurately measured the valve position. Similar results were obtained by replacing the other input signals one by one with random noise.

15.5 NEUROFUZZY APPROACHES TO ANTICIPATORY CONTROL

Anticipatory systems are systems where change of state is based on information pertaining to present as well as future states. Cellular organisms, industrial processes, and global markets provide many examples of behavior where global output is the result of anticipated, as well as current, states. In the global economy, for example, the anticipation of an oil shortage or of a significant default of foreign loans can have profound effects upon the course of the economy, whether or not the anticipated events come to pass (Holland, 1988). Participants in the economy build up models of the rest of the economy and use them to make predictions. The models are more *prescriptive* (prescribing what should be done in a given situation) than *descriptive* (describing the options of a given situation) and involve strategies appropriately formulated in terms of *lookahead*, or anticipation of market conditions. In an industrial process, the prescriptions are typically standard operating procedures (SOPs), dictating actions to be taken under specific conditions. The accumulated experience of various decision-makers at all levels of the process provides increasingly refined SOPs and progressively more sophisticated interactions amongst them and computer tools designed to assist the operators. As another example, consider a car driven on a busy highway. The driver and the car taken together are a simple, everyday example of an anticipatory system. An automobile driver makes decisions on the basis of predicting what may be happening in the future, not simply reacting to what happens at the present. Driving requires one to be aware of future system inputs by observing the curvature and grade of the road ahead, road conditions, and the behavior of other drivers. Perceptual information received at the present may be thought of as input to internal predictive models. Such a system, however, is very difficult to model using conventional approaches. In part, the difficulty relates to the fact that conventional predictive models are unduly constrained by excessive precision. Generally, in situations like the driver–car system, it is important for a decision-maker (the driver) to use a *parsimonious description* of the overall situation—that is, a model with the appropriate level of precision. Predictions about the future are not very precise, and, of course, they may be wrong. Yet, their efficacy does not rest on *precision* as much as on the more general issue of *accuracy* and their successful utilization. High levels of precision may not only be unnecessary for problems utilizing predicted values, they may very well be counterproductive. An overprecise driver may actually be a dangerous driver.

Although anticipatory systems have been studied by a number of researchers in the context of mathematical biology (Rosen, 1985), it should be noted that automata theory (Trachtenbrot and Barzdin, 1973), preview control (Tomizuka and Whitney, 1975), and their epistemological roots may be traced back to Aristotle's views on causality. It is only recently that the advent of modern computing technologies makes it possible to employ them for complex system regulation and management (Berkan et al., 1991; Tsoukalas et al., 1990, 1994b). In Japan, the automatic train operator (ATO) used in Sendai's subway system, as well as some tunnel ventilation systems and elevator control systems employ anticipatory control strategies (Yasunobu, 1985); and researchers at Tohoku University and Mitsubishi Research Institute have studied an innovative anticipatory guidance and control system for computer-assisted operation of nuclear power plants (Washio, 1993).

Probabilistic Predictions

In preview control, future information was considered as probabilistic in kind, and the control problem was seen as a problem of time delay (Tomizuka and Whitney, 1975). The situation is illustrated in Figure 15.10*a* where a discrete control problem that lasts n time steps, presently at time i, is

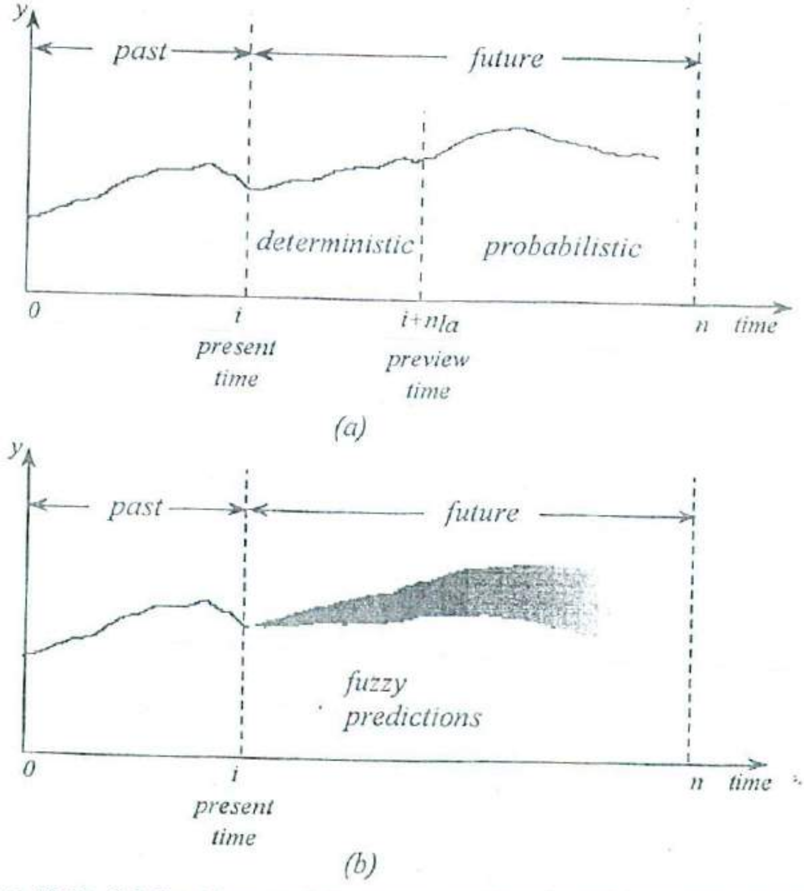


Figure 15.10 (*a*) The future in finite preview problems is modeled deterministically and stochastically. (*b*) The future in anticipatory systems is modeled fuzzily.

considered. Tomizuka postulated that up to certain time n_{la} beyond i, predictions can be made and utilized by the controller at i. Thus, the future is divided into *deterministic* and *probabilistic* parts as seen in Figure 15.10*a*. The controller is assumed to make use of preview information with respect to a command signal (desired trajectory) from the present time i up to n_{la} time units into deterministic future. The quantity n_{la} is the *preview time* (or *length of anticipation*) and is usually shorter than n, the problem duration, often by one or two time steps. To make the solution applicable to a broader class of problems, measurements of time delay, observation noise, and driving noise were included in formulating the problem. The solution showed how to utilize the local future information obtained by finite preview (n_{la}) in order to minimize an optimality criterion evaluated over the problem duration n. It was found that preview dramatically improved the performance of a system relative to nonpreview optimal performance, and a heuristic criterion about the preview time, n_{la}, was suggested, that is, $n_{la} \approx 3 \times$ (longest closed-loop plant time constant).

Fuzzy Predictions

The point of departure for our formulation is that future information is essentially *fuzzy* in nature, that is, predicted values are not imbued with stochastic or probabilistic type of uncertainty. Whatever can be said about the future does not come from measurements but instead from models; hence, such predictions are fuzzy numbers—that is, linguistic categorizations of information pertaining to the future of the system. Generally, fuzziness is a property of language, whereas randomness is a property of observation; and since there is no physical measurement pertaining to the future, the mathematics of fuzzy sets may be more appropriate for anticipatory systems. Consider, for example, the process depicted in Figure 15.10*b*. At any time i we have available information from the present as well as information from the output of some predictive model. According to our formulation, this is a fuzzy prediction. Therefore, the mathematical tools for utilizing it at time i ought to be fuzzy as well. The time Δt into the future, the anticipatory time step, depends on the nature of the problem and the predictive model used and, generally, need not be one or two time steps as is often the case in preview control. As is suggested in Figure 15.10*b*, the fuzziness of a prediction is postulated to depend on time in the future in the sense that for greater time we get fuzzier predictions.

Issues of Formalism in Anticipatory Systems

A system that makes decisions in the present on the basis of what may be happening in the future is thus envisioned to be different in two important respects: in the *language* used to formulate models of its behavior and in the method of *measurement* used to access future states. The first we call the *issue of formalism* and we address it in this section, while the latter we examined in the previous section through the concept of virtual measurement.

Consider the typical systems formulation in modern control theory. A system is described by a set of difference (differential) equations of the form

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + w(t), \qquad x(t_0) = x_0 \\ y(t) &= Cx(t) + v(t) \end{aligned} \tag{15.5-1}$$

where $\{u(t)\}$ is an $r \times 1$ input sequence. $\{y(t)\}$ is an $m \times 1$ output sequence, $\{x(t)\}$ is an $n \times 1$ state sequence, A, B, and C are appropriate transition matrices, x_0 some initial state, and, $w(t)$ and $v(t)$ are noise terms.

A system is called *anticipatory* if $x(t+1)$ and $y(t)$ are not uniquely determined by $x(t)$ and $u(t)$ alone, but use information pertaining to some future state $x(t+\Delta t)$ and/or input $u(t+\Delta t)$.

Looking at equations (15.5-1) we observe that it is rather difficult to include future information in these equations except by containing it within the noise terms as in the case of nondeterministic systems. In such a case, one obtains sets of values $x(t+1)$ and $y(t)$ with each pair $[x(t), u(t)]$. Suppose that the values of x and y are subsets of some larger sets X and Y. If we denote these subsets of X and Y by X^{t+1} and Y^t, we obtain mappings of the form

$$\begin{aligned} X^{t+1} &= F[x(t), u(t)] \\ Y^t &= G[x(t), u(t)] \end{aligned} \tag{15.5-2}$$

Of course, one could fuzzify this system by assuming that these X^{t+1} and Y^t are fuzzy subsets on X and Y, respectively, and obtain a fuzzy system determined by conditional membership functions

$$\begin{aligned} &\mu[x(t+1) \,|\, x(t), u(t)] \\ &\mu[y(t) \,|\, x(t), u(t)] \end{aligned} \tag{15.5-3}$$

Subsequently the compositional rule of inference may be used to calculate the fuzzy response of the fuzzy system to any fuzzy input. The problem, however, of involving future information in the formulation of equations (15.5-1) still remains. Generally, if we do so, the mappings in equations (15.5-2) cease to be *many-to-one* mappings (i.e., *functions*) but instead become more general *many-to-many* mappings such as we now have in *fuzzy relations*.

Consider again equations (15.5-1). Another approach is to consid [illegible] he equal signs " = " as the assignment operators " := ", that is

$$
\begin{aligned}
x(t+1) &:= Ax(t) + Bu(t) + w(t), \qquad x(t_0) = x_0 \\
y(t) &:= Cx(t) + v(t)
\end{aligned}
\tag{15.5-4}
$$

where the assignment operator " := " is an *if/then* rule, which assigns the right-hand side (RHS) of equation (15.5-4) to the left-hand side (LHS) upon update. Now we are in the realm of logical implications and we can easily include terms such as $x(t + \Delta t)$, and $u(t + \Delta t)$ in our *if/then* rules. The calculus of fuzzy *if/then* rules is rather well known and provides an interesting alternative and enhancement of formulations such as equation (15.5-1), particularly for the purpose of qualitative and complex system modeling. Thus, an anticipatory system can be described by a collection of fuzzy *if/then* rules

$$
R^N = \{R^1, R^2, \ldots, R^n\} \tag{15.5-5}
$$

Each rule is a *situation/action* pair, denoted as $s \rightarrow a$, where both *present* and *anticipated situations* are considered in the LHS and *current action* is considered in the RHS. The rules of equation (15.5-5) may be rewritten as

$$
\begin{aligned}
R^N &= \{s^1 \rightarrow a^1, s^2 \rightarrow a^2, \ldots, s^n \rightarrow a^n\} \\
&= \mathop{\phi_\alpha}_{j=1}^{n} (s^j \rightarrow a^j)
\end{aligned}
\tag{15.5-6}
$$

where ϕ_α is an appropriate implication operator (Terano et al., 1992). In many cases we can further partition the set of rules in equation (15.5-6) into rule bases (RB), with each rule base being responsible for one action; that is,

$$
R^N = \bigcup_{j=1}^{r} [RB^p] \tag{15.5-7}
$$

Rule bases (15.5-7) can be made to reflect temporal partitions that is, we can have rules that describe the state of the system at t, that is,

$$
s(t) \rightarrow a(t) \tag{15.5-8}
$$

and we can also have rules that describe the possible state of the system at some time later, that is,

$$
s(t + \Delta t) \rightarrow a(t) \tag{15.5-9}
$$

Thus an anticipatory fuzzy algorithm can infer the current action $a(t)$ on the basis of the present state $s(t)$ as well as anticipated ones $s(t + \Delta t)$.

Generally, the rules of (15.5-5) describe relations of a more general type than that of functions, i.e., *many-to-many* mappings (see Chapter 5). Such mappings have the linguistic form of fuzzy *if/then* rules—for example,

$$\text{if} \quad x \text{ is } A \quad \text{then } y \text{ is } B \tag{15.5-10}$$

where x is a fuzzy variable whose arguments are fuzzy sets denoted as A, and y is a fuzzy variable whose arguments are the fuzzy sets B. Similar rules pertaining to future states are of the form

$$\text{if} \quad x \text{ will be } A, \quad \text{then } y \text{ is } B \tag{15.5-11}$$

where x is thought of as a situation variable and y is the corresponding action variable. Evaluation of formulations using rules such as (15.5-10) and (15.5-11) can be done through *generalized modus ponens* as we have seen in Chapter 5.

Anticipatory control strategies may be based on global fuzzy variables such as *performance* where a decision at each time t is taken in order to maximize current as well as anticipated performance pertaining to $t + \Delta t$. *Performance* in this case is a fuzzy variable (with an appropriate set of fuzzy values) that summarizes information about the system, thereby allowing the system to make decisions about its change of state. The observation/prediction of such variables can be addressed by the methodology presented in Section 15.4.

Alternatively we may use fuzzy *if/then* rules to generate a decision from (15.5-7) and call a predictive routine to anticipate the effect of the proposed decision on the system output (Yasunobu and Miyamoto, 1985). Additional rules may be called if the current decision will result in system behavior which is unacceptable. Consider, for example, the following rule:

If the current decision (u_c) *will cause the difference between the current and anticipated states to be big, then*

$$u = u_c(1 - \beta \cdot bigt) \tag{15.5-12}$$

where β is a user-chosen parameter between 0 and 1 and *bigt* is the fulfillment function for the anticipated difference states. The parameter β may also be chosen by employing a predictive neural network (McCullough, 1993).

REFERENCES

Alguidingue, I. E., Loskiewicz-Buczak, A., and Uhrig, R. E., Clustering and Classification Techniques for the Analysis of Vibration Signatures, *Applications of Artificial Neural Networks III*, Proceedings of the SPIE, Orlando, FL, April 20–24, 1992.

Alguindigue, I. E., Loskiewicz-Buczak, A., and Uhrig, R. E. Monitoring and Diagnosis of Rolling Element Bearings using Artificial Neural Networks, *IEEE Transactions* *on Industrial Electronics: Special Issue on Applications of Intelligent Systems to Industrial Electronics*, April 1993.

Berkan, R. C., Upadhyaya, B. R., Tsoukalas, L. H., Kisner, R. A., and Bywater, R. L, Advanced Automation Concepts for Large-Scale Systems, *IEEE Control Systems*, Vol. 11, No. 6, pp. 4–12, 1991.

Berry, J., *Tracking of Rolling Element Bearing Failure Stages Using Vibration Signature Analysis*, Technical Associates of Charlotte, Inc., 1990.

Broch, J. T., *Mechanical Vibrations and Shock Measurements*, Bruel & Kjaer, Naerum, Denmark, 1984.

Djumovic, D. Weighted Conjunctive and Disjunctive Means and Their Application in System Evaluation, *Publikacije Elektrotehnickog Faculteta Beograd, Serija Matematika i Fizika*, No. 483, 1974.

Dyckhoff, H., and Pedrycz, W., Generalized means as Model of Compensative Connectives, *Fuzzy Sets and Systems*, Vol. 14, 1984.

Hewlett Packard, *Effective Machinery Maintenance Using Vibration Analysis*, Application Note 243-1, Dynamic Signal Analyzer Applications, 1983.

Holland, J. H., The Global Economy as an Adaptive Process, in *The Economy as an Evolving Complex System*, P. W. Anderson, K. J. Arrow, D. Pines, Eds. A Proceedings Volume in the Santa Fe Institute in the Sciences of Complexity, Addison-Wesley, Reading, MA, 1988.

Ikonomopoulos, A., Tsoukalas, L. H., and Uhrig, R. E., Integration of Neural Networks with Fuzzy Reasoning, for Measuring Operational Parameters in a Nuclear Reactor, *Nuclear Technology*, Vol. 104, pp. 1–12, 1994.

Ikonomopoulos, A., Uhrig, R. E., and Tsoukalas, L. H., A Methodology for Performing Virtual Measurement in a Nuclear Reactor System, *Transactions of American Nuclear Society* 1992 *Winter Meeting*, Chicago, IL, November 15-20, 1992, pp. 106–109.

Jackson, C., *The Practical Vibration Primer*, Gulf Publishing Company, Houston, TX, 1979.

Kaufmann, A., and Gupta, M. M., *Introduction to Fuzzy Arithmetic*, Van Nostrand Reinhold, New York, 1991.

Kohonen, T., The Self-Organizing Map, *Proceedings of the IEEE*, Vol. 78, No. 9, 1990.

Krishnapuram, R., Lee, J., Fuzzy-Connective-Based Hierarchical Aggregation Networks for Decision Making, *Fuzzy Sets and Systems*, Vol. 46, pp. 11–27, 1992.

Loskiewicz-Buczak, A, and Uhrig, R. E., Probabilistic Neural Network for Vibration Data Analysis, in *Intelligent Engineering Systems Through Artificial Neural Networks*, Vol. 2, ASME Press, New York, 1992.

Loskiewicz-Buczak, A., and Uhrig, R. E., Aggregation of Evidence by Fuzzy Set Operations for Vibration Monitoring, *Proceedings of the Third International Conference on Industrial Fuzzy Control & Intelligent Systems*, R. Langari, J. Yen, and J. Painter, eds., Houston, TX, December 1–3, 1993a.

Loskiewicz-Buczak, A., and Uhrig, R. E., Neural Network—Fuzzy Logic Diagnosis Systems for Vibration Monitoring, *Proceedings of ANNIE-93, Artificial Neural Networks in Engineering Conference*, St. Louis, MO, November 14–17, 1993b.

Loskiewicz-Buczak, A., and Uhrig, R. E., Decision Fusion by Fuzzy Set Operation, in *Proceedings of IEEE World Congress on Computational Intelligence, Fuzzy Systems Conference*, Orlando, F. L., June 26–30, 1994, pp. 1412–1417.

Loskiewicz-Buczak, A., Alguindigue, I., and Uhrig, R. E., Monitoring and Diagnosis of Rolling Element Bearings Using Artificial Neural Networks, in *IEEE Transactions on Industrial Electronics: Special Issue on Applications of Intelligent Systems to Industrial Electronics*, April 1993b.

Loskiewicz-Buczak, A., Alguindigue, I. E., and Uhrig, R. E., Vibration Analysis in Nuclear Power Plants Using Neural Networks, in *Proceedings of Second International Conference on Nuclear Engineering*, San Francisco, CA, March 21–24, 1993a.

McCullough, C. L., Anticipatory Neúro-Fuzzy Control: A Powerful New Method for Real World Control, in *Proceedings of IEEE International Workshop on Neuro Fuzzy Control*, Muroran, Japan, March 22–23, 1993, pp. 267–272.

NeuralWare, Neural Computing-NeuralWorks Professional II/PLUS and NeuralWorks Explorer, *User Manual*, NeuralWare, Inc., Pittsburgh, 1991.

Rosen, R., *Anticipatory Systems*, Pergamon Press, New York, 1985.

Sugeno, M., and Yasukawa, T., A Fuzzy-Logic-Based Approach to Qualitative Modeling, *IEEE Transactions on Fuzzy Systems*, Vol. 1, No. 1, pp. 7–31, 1993.

Terano, T., Asai, K., and Sugeno, M., *Fuzzy Systems Theory and Its Applications*, Academic Press, Boston, 1992.

Tomizuka, M., and Whitney, D. E., Optimal Finite Preview Problems (Why and How Is Future Information Important), *Journal of Dynamic Systems, Measurement, and Control*, pp. 319–325, December 1975.

Trachtenbrot B. A., and Barzdin Y. M., *Finite Automata Behavior and Synthesis*, North-Holland, Amsterdam, 1973.

Trivedi, M. M., and Bezdeck, J. C., Low-Level Segmentation of Aerial Images with Fuzzy Clustering, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-16, No 4., 1986.

Tsoukalas, L. H., Ikonomopoulos, A., and Uhrig, R. E., Generalized Measurements in Neuro-Fuzzy Systems 1993, in *Proceedings of the International Workshop on Neural Fuzzy Control*, Muroran, Japan, March 22–23, 1993.

Tsoukalas, L. H., Berkan, R. C., and Ikonomopoulos, A., A Methodology for Uncertainty Management in Knowledge-Based Systems Employed Within the Anticipatory Paradigm, in *Proceedings of the Third International Conference of Information Processing and Management of Uncertainty in Knowledge-Based Systems*, IPMU, Paris, July 2–6, 1990, pp. 77–80.

Tsoukalas, L. H., and Ikonomopoulos, A., Uncertainty Modeling in Anticipatory Systems, in *Analysis and Management of Uncertainty*, B. M. Ayyub, M. M. Gupta and L. N. Kanal eds., Machine Intelligence and Pattern Recognition Series, Elsevier/North Holland, Amsterdam, 1992, pp. 79–91.

Tsoukalas, L. H., Ikonomopoulos, A., and Uhrig, R. E., Fuzzy Neural Control, in *Artificial Neural Networks for Intelligent Manufacturing*, C. H. Dagli, ed., pp 413–434, Chapman & Hall, London, 1994.

Tsoukalas, L. H., Ikonomopoulos, A., and Uhrig, R. E., Neuro-Fuzzy Approaches to Anticipatory Control, in *Artificial Intelligence in Industrial Decision Making, Control* *and Automation*, S. G. Tzafestas and H. B. Verbruggen eds., Kluwer Academic Publishers, The Netherlands, 1995, pp. 405–419.

Washio, T., and Kitamura, M., General Framework for Advance of Computer-Assisted Operation of Nuclear Plants—Anticipatory Guidance and Control for Plant Operation, *Proceedings of Japan Atomic Energy Society*, Kobe, Japan, October 1993 (in Japanese).

Yasunobu, S., and Miyamoto, S., Automatic Train Operation by Predictive Fuzzy Control, in *Industrial Applications of Fuzzy Control*, M. Sugeno, ed., pp. 1–18, North Holland, Amsterdam, 1985.

Zadeh L. A., Fuzzy Sets, *Information and Control*, Vol. 8, 1965.

Zimmermann, H. J., *Fuzzy Sets, Decision Making, and Expert Systems*, Kluwer Academic Publishers, Boston, 1987.

Zwingelstein G., and Hamon, L., EDF Studies on the Condition Monitoring of Rolling Element Bearings, in *Proceedings of the 4th Incipient Failure Detection Conference*, Philadelphia, October 1990.

IV

OTHER ARTIFICIAL INTELLIGENCE SYSTEMS

16

EXPERT SYSTEMS IN NEUROFUZZY SYSTEMS

16.1 INTRODUCTION

Artificial intelligence is a branch of computer science that attempts to emulate certain mental processes of humans by using computer models. In expert systems, perhaps the first field of artificial intelligence to be commercially recognized in its own right, one of the primary objectives is to mimic human expertise and judgment using a computer program by applying knowledge of specific areas of expertise to solve finite, well-defined problems. These computer programs contain human expertise (called *heuristic knowledge*) obtained either directly from human experts or indirectly from books, publications, codes, standards, or databases, as well as general and specialized knowledge that pertains to specific situations. Expert systems have the ability to reason using formal logic, to seek information from a variety of sources including databases and the user, and to interact with conventional programs to carry out a variety of tasks including sophisticated computation.

The principal use of expert systems in neurofuzzy systems is to ensure that the unique capabilities of neural networks and fuzzy logic systems are implemented in the proper way including the fuzzy rules of fuzzy algorithms, sometimes called "fuzzy associate memory" (FAM) matrices (see Figure 6.2). While it may be possible to use ordinary software programs to do this, the ability of expert systems to adapt to and deal with unforeseen situations is very important when the outputs may not be precise or the model may be based on less-than-perfect data.

16.2 CHARACTERISTICS OF EXPERT SYSTEMS

A number of characteristics of expert systems are unique and generally advantageous [see, for example, Van Horn (1986) and Feigenbaum et al., (1988)]:

1. Experts need not be present for a consultation; expert systems may be delivered to remote locations where expertise may not be otherwise available.
2.· Expert systems do not suffer from some of the shortcomings of human beings (e.g., they do not get tired or careless as the work load increases) but, when properly used, continue to provide dependable and consistent results.
3. The techniques inherent in the technology of expert systems minimize the recollection of information by requesting only relevant data from the user or appropriate databases (i.e., data encountered in the reasoning path).
4. Expert knowledge is saved and readily available because the expert system can become a repository for undocumented knowledge that might otherwise be lost (e.g., through retirement).
5. The development of expert systems forces documentation of consistent decision-making policies. The clear definition of these policies makes the overall decision-making process transparent and the implementation of policy changes instant and simultaneous at all sites.

On the other hand, expert systems have disadvantages that affect their use:

1. They usually deal only with static situations.
2. They must be kept up to date as conditions change.
3. They often cannot be used in novel or unique situations.
4. Results are very dependent on the adequacy of the knowledge incorporated into the expert system.
5. Perhaps most important, they do not benefit from experience except through updating of the knowledge base (based on human experience).
6. Expert systems are unable to solve problems outside their domain of expertise. In many cases they are unable to detect the limitations of their domain (Swartout and Smoliar, 1987; Ricker, 1986).

The domain of an expert system refers to the scope of the knowledge contained within the knowledge base. If the expert system operates outside its domain, it is possible that it may generate incorrect results by utilizing nonapplicable, irrelevant knowledge while searching for a solution. The inability of expert systems to recognize the limitations of their knowledge has been identified as a very serious shortcoming.

Expert systems can, under certain circumstances, deal with imprecise or "fuzzy" information, missing information, and even a certain amount of conflicting information through the use of "certainty factors" or Bayesian probabilities (Kaplan et al., 1987). Certainty factors represent a measure of belief of the user that a piece of evidence is true. These are not probabilities but rather simply a subjective judgment on the degree of truth or validity of an assertion. Some of the information used in development and application of an expert system may not be absolutely certain, and the use of certainty factors allows this subjective evaluation to be incorporated into the expert system. The final results in these cases may be the "most probable" solution or the "best" solution, but there is no absolute guarantee that the solution is the "correct" solution. Recent work incorporating "fuzzy logic" and "reasoning under uncertainty" into expert systems has greatly improved the performance of expert systems when dealing with complex systems.

A comparison of human and artificial expertise will help convey the strengths and weaknesses of expert systems. Human expertise is perishable and difficult to transfer, whereas artificial expertise is permanent and easy to transfer. Human expertise is not always consistent, whereas artificial expertise is consistent. (If you give an expert system the same problem on two occasions you will get the same answer unless stochastic processes are involved; this is not necessarily true of a human expert.) On the other hand, human expertise is creative and has a broad focus, whereas artificial expertise is uninspired and usually has a very narrow focus. Above all, human expertise is adaptive and demonstrates common sense, characteristics usually lacking in expert systems because the knowledge is entirely technical or objective in nature. For instance, artificial expertise does not know that the objects cannot occupy the same space unless it is told.

16.3 COMPONENTS OF AN EXPERT SYSTEM

The principal components of an expert system are the *inference engine*, the *knowledge base*, and the *interface* between the expert system and humans (users, knowledge engineers, and experts). The inference engine is a computer program that gathers the information needed from the knowledge base, associated databases, or the user, guides the search process in accordance with a preselected strategy, uses rules of logic to draw inferences or conclusions for the processes involved, and presents these inferences or conclusions (where warranted) with explanations or bases.

The knowledge base consists of information stored in retrievable form in the computer, usually in the form of rules or frames. The correctness and completeness of the information within the knowledge base is the key to obtaining correct results or solutions using expert systems. Knowledge bases may contain models of systems which produce real-time results or certain learning systems (such as neural networks) that provide new knowledge.

The interface between the human and the expert system must translate user input into the computer language, and it must present conclusions and explanations to the user in clear written or graphical form. It should also include an editor to assist in adding to or changing the knowledge base.

One of the major breakthroughs in development of expert systems came in the mid-1970s with the expert system MYCIN, a diagnostic system for infectious diseases of the blood. The MYCIN architecture completely separated the knowledge base from the inference engine, which permitted modification of the knowledge base without any influence on the inference engine. Hence, it was possible to start with a simple expert system and incrementally add features and complexity as needed. Such a separation is common today even in conventional software, but it was a significant advancement in the mid-1970s.

The knowledge base of an expert system contains the expertise (facts and heuristics) collected from experts, books, publications, and other sources and encoded into rules, frames, or other computer representations of knowledge. This information describes a methodology for solving the problem as a human expert would solve it. Collecting adequate knowledge from experts and translating it into computer code (a process called "knowledge acquisition") has proven to be a very difficult task. All too often, experts really do not understand the processes by which they reason or solve problems. In other cases, experts are reluctant to give up their expert knowledge because they perceive that the availability of an expert system with their expertise may lessen their value to their employer or clients. Because an expert system is only as good as its knowledge base, proper collection and representation of knowledge is critical for the successful implementation and operation of expert systems.

Some expert systems contain a degree of self-awareness or self-knowledge that allows them to reason about their own operation and to display inference chains and traces of the rationale behind their results (Waterman, 1986). These abilities (the explanation facilities) have been recognized as one of the most valuable features of expert systems. The user can take advantage of explanation facilities to request a complete trace for a consultation, request an explanation on how a particular goal or subgoal was inferred, or request an explanation of why a particular piece of information is needed. These facilities can be used to obtain information on the status of a system. Explanation-generating facilities are also of great use in debugging expert systems and may play a key role in verification and validation of expert systems.

The performance of mature expert systems has shown that the reliability of an expert system in a given subject area asymptotically approaches the reliability of the expert as the knowledge base approaches the expert's knowledge in that area. In some cases the reliability of an expert system exceeds the reliability of the expert, not because the expert system is "smarter" than the expert, but rather because the expert system does not forget anything contained in the knowledge base and is capable of rapidly carrying out analytic and mathematical operations.

An expert system "shell" is a computer program used to develop an expert system. Early shells were expert systems from which the domain-specific knowledge bases had been removed and the mechanism for creating a new knowledge base of the user's choice had been made "user friendly." Often a shell also has provisions for changing the reasoning processes of the inference engine to adapt to the specific problem. The first shell was EMYCIN (essential MYCIN), in which the knowledge base on infectious diseases of the blood was removed from MYCIN and knowledge bases on cancer treatment and pulmonary diseases were used to create two new expert systems (ONCONIN and PUFF, respectively) to assist doctors in these fields. The pioneering efforts of Stanford University on EMYCIN paved the way for virtually all modern expert system shells. Indeed, only in the last few years have expert system shells begun to deviate significantly from the overall structure developed for MYCIN.

Expert system shells today differ significantly from each other and offer the user a wide variety of capabilities. Some have sacrificed size of knowledge base to improve ease of updating the knowledge base, and vice versa. Certain expert systems (e.g., 1ST CLASS and VP EXPERT) have the ability to derive the knowledge base from a series of examples by induction. Such ability to extract information from databases and experimental results are one of the strengths of artificial neural networks. Hence, the use of a hybrid consisting of an artificial neural network in the knowledge base of an expert system is feasible and often advantageous. Recently, an expert system shell was introduced with HYPERTEXT as part of the knowledge base. Selection of an expert system to fit a specific need is almost a research project in itself and has, in fact, been the topic for an expert system.

16.4 KNOWLEDGE REPRESENTATION AND INFERENCE

There are a variety of approaches to encode human expertise in expert systems, the most common one being *if/then* rules. Semantic networks, frames, and logical expressions are alternative paradigms of knowledge representation, although the majority of industrial expert systems use the rule-based paradigm. [For a discussion of the subject see (Gonzalez and Dankel, 1993).)

The three basic constituents of a rule-based expert system are: *rule base*, *working memory* and *rule interpreter*. The rule base is often partitioned into groups of rules, called *rule clusters*. Each rule cluster encodes the knowledge required to perform a certain *task* or a fraction of a task, usually referred to as a *subtask*. There may also be rules for internal control purposes—for example, to signal which rule cluster to select as holding potentially relevant knowledge at a given time. Collectively, these rules are referred to as the *control structure* of an expert system. Another class of rules, called *demons*, may be present; they are designed to function outside the control structure of the program for the purpose of enhancing its ability to respond quickly to the occurrence of an event requiring some immediate action. Demons address inefficiency issues that may arise from excessive control over the rule base (Cooper and Wogrin, 1988).

Working memory is a database holding input data, inferred hypotheses, and internal information about the program. In an on-line expert system with monitoring functions, for example, the state of working memory at any given time reflects changes occurring in the process being monitored as well as internal changes due to the reasoning process of the program itself.

The mechanism through which rules are selected to be fired is called the *rule interpreter*. It is based on a pattern matching algorithm whose main purpose is to associate at any given time the state of the system (input data, inferred hypotheses, etc.) with applicable rules from the rule base.

The inference engine of an expert system is in charge of manipulating the data presented to the system and arriving at a conclusion. In expert system technology the two most widely used reasoning techniques are forward chaining (forward reasoning) and backward chaining (backward reasoning). In forward chaining the system reasons forward from a set of known facts and tries to infer the conclusions or goals. Design of a complex system is a forward-chaining application where the expert system starts with the known requirements, investigates the very large array of possible arrangements, and makes a recommendation based on criteria specified by the user.

In backward chaining the system works backward from tentative conclusions or goals and attempts to find supporting evidence to verify their correctness. Solving a crime is a backward chaining application where the expert system identifies the possible suspects, looks for evidence indicating the guilt and innocence of each suspect, and makes a recommendation regarding which suspect is the most likely criminal. In many cases, backward-chaining systems are more efficient than true forward-chaining systems because they tend to reduce the search space and arrive at a conclusion more quickly.

Many advanced expert systems use a combination of both forward and backward chaining. Different search strategies, such as "depth first" or "breadth first", may be incorporated into either backward or forward chaining.

Data enter an expert system either through a user interface or from other programs such as databases, data acquisition systems, simulation packages, and so on, and form the initial facts (or assertions or evidence) available to the rules. From the input data, conclusions are drawn in a process called *inferencing*. The two basic inferencing strategies, *forward chaining* and *backward chaining*, are also referred to as *modus ponens* and *modus tollens*, respectively. These are crisp versions of GMP and GMT introduced in Chapter 5 for fuzzy systems.

In *modus ponens*, if we have the following rule

$$\text{if} \quad A \text{ is true}, \quad \text{then } B \text{ is true}$$

Hence if it is known that "A is true," then we can infer that "B is true." Most expert systems use this powerful inferencing strategy. In *modus tollens*, if we know that the rule is true and we also know that "B is false," then we can infer that "A is false." We often simply write A instead of "A is true" and NOT A instead of "A is false." The requirement for an exact match between input data and what is stated in a rule is relaxed in fuzzy expert systems where fuzzified versions of the basic inferencing strategies have been developed.

16.5 UNCERTAINTY MANAGEMENT

An important issue in expert systems is uncertainty management. Representing or combining uncertain data and drawing reliable inferences from it has been extensively investigated over the past two decades, and several theories of uncertainty have provided tools for solving uncertainty problems (Kruse et al., 1991). Probability theory and certainty factors have been frequently used; but also possibility (fuzzy) theory, Dempster–Shafer belief measures, Cohen's theory of endorsements, and subjective Bayesian methods are uncertainty management paradigms that have found important applications.

The oldest approach to uncertainty management has been the probabilistic approach which essentially ascribes probabilities to facts and rules and uses Bayes' rule and a rather large amount of statistical data to construct the various probabilities in the knowledge base (Kruse et al., 1991). A drawback of the probabilistic approach is its difficulty in distinguishing between *absence of belief* and *doubt* or to represent how *ignorance* is related to the lack of knowledge.

Certainty Factors

In order to overcome the difficulties of Bayesian probabilities (e.g., requiring a large volume of data or distinguishing between *absence of belief* and *doubt*), certainty factors may be used. In the certainty factor (CF) formalism, knowledge is expressed as a set of rules having the form

$$\text{if} \quad E, \quad \text{then } H \quad \text{with } CF(H|E)$$

where E is the *evidence*—that is, one or more facts known to support the *hypothesis* H, and $CF(H|E)$ is the certainty factor for the rule, a measure of belief in H, given that E has been observed. The value of CF ranges from -1 to $+1$. When $CF = -1$ the hypothesis H is totally denied, while at $CF = +1$, the hypothesis H is totally confirmed.

Certainty factors are obtained from *measures of belief*, $MB(H, E)$, and *measures of disbelief*, $MD(H, E)$, both taking values between 0 and 1. A measure of belief $MB(H, E)$ represents the degree to which the belief in hypothesis H is supported by observing evidence E, and it is computed by

$$MB(H, E) = \begin{cases} 1 & \text{if } p(H) = 1 \\ [p(H|E) - (H)]/[1 - p(H)] & \text{else} \end{cases} \qquad (16.5\text{-}1)$$

A *measure of disbelief* $MD(H, E)$, on the other hand, represents the degree to which the disbelief in hypothesis H is supported by evidence E. It is computed by

$$MD(H, E) = \begin{cases} 1 & \text{if } p(H) = 1 \\ [p(H) - p(H|E)]/[1 - p(H)] & \text{else} \end{cases} \qquad (16.5\text{-}2)$$

The certainty factor CF is defined in terms of $MB(H, E)$ and measure of disbelief $MD(H, E)$

$$CF = [MB(H, E) - MD(H, E)]/\{1 - \text{MIN}[MB(H, E), MD(H, E)]\} \qquad (16.5\text{-}3)$$

During the execution of a knowledge base, multiple rules are typically capable of deriving the same hypothesis or conclusion, resulting in modification of the CF's involved. Consider, for example, a case where two different evidences E_1 and E_2 lead to the same hypothesis H. In such cases, certainty factors of the same or opposite signs can be combined directly by the following formulas (Gonzalez, 1993; Kruse et al., 1991):

Case 1. When both $CF(H|E_1)$ *AND* $CF(H|E_2) > 0$

$$CF(H|E_1, E_2) = CF(H|E_1) + CF(H|E_2) - CF(H|E_1) * CF(H|E_2)$$

Case 2. When $-1 < CF(H|E_1) * CF(H|E_2) \leq 0$

$$CF(H|E_1, E_2)$$
$$= [CF(H|E_1) + CF(H|E_2)]/\{1 - \text{MIN}[|CF(H|E_1)|, |CF(H|E_2)|]\}$$

Case 3. When $CF(H|E_1)*CF(H|E_2) = -1$

$$CF(H|E_1, E_2) = \textit{undefined}$$

Case 4. When both $CF(H|E_1)$ *AND* $CF(H|E_2) < 0$

$$CF(H|E_1, E_2) = CF(H|E_1) + CF(H|E_2) + CF(H|E_1)*CF(H|E_2)$$

It has been assumed in the above equations that we have absolute confidence in the evidence of premises used to derive various values. In expert systems, often (but not always) a hypothesis from a rule is used as evidence for another rule and hence we should not actually have absolute confidence in the evidence, and the certainty factor approach does not materially contribute to the final results. An additional drawback of certainty factors seems to be the complexity of maintaining them. When, for example, new knowledge is added or deleted from the knowledge base, the certainty factors of existing knowledge change as well, making the maintenance of the system rather complicated. For these reasons and others, use of fuzzy set theory in the form of reasoning under uncertainty is more commonly encountered today.

16.6 STATE OF THE ART OF EXPERT SYSTEMS

The impact of expert systems technology has been felt in many areas of science, education, and industry. In the past decade a great many applications have been initiated, and many are now operational or in the prototype stage. (Uhrig, 1988; Hertz, 1988). The extent of the potential application of this technology is not yet known, because expert systems in the future may be used in completely new settings to solve quite different problems. However, the introduction of fuzzy rules has greatly enhanced the usefulness of expert systems.

It is very difficult to gain a true picture of just how widespread the use of expert systems has become. In many cases, organizations are using expert systems internally. Even the fact that they are used, let alone the details of the expert systems, are treated as proprietary for the simple reason that the company or organization wants to gain competitive advantage. By one analyst's estimate, about half of the companies listed in the Fortune 500 are developing expert systems (Coates, 1988). One automobile manufacturer is reportedly insisting that manufacturers supply diagnostic expert systems with the equipment they provide.

Expert systems may change the manner in which many organizations operate, and they could change the workplace in general. In large organizations such as government, big corporations, and associations, one expert predicts that 60–90% of all jobs are candidates for augmentation, displacement, or replacement by expert systems (Coates, 1988). Coates further predicts that by about the turn of the century the capabilities of expert systems will have grown to such a degree that their impact will be felt throughout most occupations and workplaces.

16.7 USE OF EXPERT SYSTEMS

Generally, but not always, problems that are amenable to a numerical solution should be solved using conventional computer programs. However, there are many situations in which expert systems offer unique advantages over conventional programs. Most applications of expert systems today can be classified into the following six categories: (1) monitoring systems, (2) control systems, (3) configuring systems, (4) planning systems, (5) scheduling systems, and (6) diagnostic systems.

Monitoring Systems

Monitoring systems are dedicated to data collection and analysis over a period of time. The collected values are compared against expected performance, and if discrepancies are identified the expert system generates recommendations and/or notifies the operator.

Control Systems

Control systems are monitoring systems in which action (e.g., opening a valve, adjusting a bias, turning on a heater, etc.) is taken as a result of the discrepancy identified by the monitoring system.

Configuring Systems

Configuring systems address problems in which a finite set of components is to be arranged in one of many possible patterns. The classical example in this category is XCON, an expert system used by a large computer manufacturer to configure its equipment in accordance with its own rules and the user specifications.

Scheduling and Planning Systems

Scheduling and planning expert systems coordinate the capabilities or components within an organization to optimize production and/or increase efficiency. The difference between planning and scheduling systems is that the components for a task are not always known in planning systems.

Diagnostic Systems

Diagnostic systems observe and analyze data and map the analysis results to a set of problems. Once the problems have been identified, the expert system usually recommends a solution based on facts in its knowledge base and on the other information it can acquire. Expert systems have been used to solve many different problems in a variety of fields. Some of these areas are listed in Table 16.1, which is intended to give a brief overview of the breadth of applications that has developed. One area in which there has been extensive efforts to utilize expert systems is the nuclear power field, many of which could affect safety and safety-related systems. The scope of these applications has been documented by Bernard and Washio (1989).

Table 16.1 Applications of expert systems

FIELD	*USE*
Design and engineering	Collecting and storing knowledge of best designers speeding the design process
Computer applications	Configuring equipment to user specifications Diagnosing problems with computer equipment
Manufacturing	Managing human and machine resources Facilitating factory automation
Finance	Decision support tools Providing tax and other business advice Processing loan and mortgage applications Analyzing financial risk
Science and medicine	Providing medical advice in hospitals Providing diagnostic assistance to medical personnel Patient monitoring
Geological applications	Advising regarding mineral deposit and oil locations Advising drillers regarding stuck bits
Training	Interface for computer-aided instruction Assisting in computer-based training

16.8 EXPERT SYSTEMS USED WITH NEURAL NETWORKS AND FUZZY SYSTEMS

Neural Network in the Knowledge Base of an Expert System

Neural networks, in spite of the extraordinary usefulness, have relatively limited capabilities. They are trained using available data, tested, and put into use. All they can do is recall an output when presented with an input consistent with the training data. They cannot reason, seek data from available databases to assist their operation, or provide an explanation of their outputs. They need a structured environment in which to operate, which can be provided in some cases by conventional software programming. However, recent experience indicates that usefulness of a neural network can be enhanced significantly if an expert system is used to provide this operating environment. Indeed, an expert system can retrain a neural network to adapt this hybrid system to new situations, or it can intermittently update the training of the neural network to adapt to changing situations. Some recent work indicates that expert systems can be used to provide explanations for why a neural network gives the output it does.

Perhaps the most direct combination of these two artificial intelligence technologies is the use of a neural network in the knowledge base of an expert system. This gives the expert system the ability to learn from data presented to it. The training may be on-line or performed during an initialization period. Multiple and/or modular neural networks may be incorporated into the knowledge base, and neural network outputs may be combined within the knowledge base. Control of the neural network is carried out by the inference engine in the same way that it seeks additional information from a database or initiates a logic reasoning step.

Fuzzy Rules in the Knowledge Base

One of the most popular methods of storing information in the knowledge base is through the use of *if/then* rules. Both the antecedent and the consequent or action of the rules may have multiple statements connected by conjunctions such as *AND* and/or *OR*. For simple systems, the rules can be relatively simple and straightforward. If the individual components of a system are independent and follow a "logic tree" structure, the rules proceed in a monotonic manner; that is, the inferencing process always proceeds forward. However, if the components are interconnected, the logic trees interact with the result that the rules become very long (more qualifying conditions connected by conjunctions), more complex, and more numerous. Hence, it is increasingly harder to prevent rules from conflicting with each other. Indeed, it has been the experience of many investigators that when the number of rules gets beyond about 200, it is virtually impossible to write a meaningful rule that does not conflict with previously written rules. This paralysis of the knowledge base for complex systems caused interest in expert systems to decline in middle to late 1980s. With the advent of fuzzy rules, based on fuzzy set technology, expert systems are again being introduced in high-technology systems. For instance, an autonomous navigation system using sensor signals to navigate between moving objects was almost abandoned when 450 rules did not provide a satisfactory system. However, the replacement of the navigation system's 450 rules with 15 fuzzy rules provided a system with outstanding performance (Pín, 1992). Comparable results in the reduction in size of expert system knowledge bases by the introduction of fuzzy rules have been reported by many investigators (Terano et al., 1994).

It is this use of fuzzy rules in an expert system, a combination that is often called "fuzzy expert systems," that has reawakened interest in expert systems. In a traditional expert system, the number of rules necessary to unambiguously define a situation tended to grow in an exponential-like fashion as the complexity of the system increased. For complex problems that were amenable to monotonic reasoning (i.e., the reasoning proceeded forward directly toward a goal with a reversal), a large number of rules simply meant a slow and cumbersome process. For complex problems that involved searching many paths with reversals and many-to-many mappings, use of a rule-based knowledge base was simply not feasible. The ability of fuzzy rules to drastically reduce the number of rules has been the secret of success in using expert systems in most complex situations.

Even in fuzzy expert systems, a major effort must be made to minimize the number of rules without deteriorating the operation. Consider the case where there are three inputs and one output that utilize five membership functions each. This could lead to 5^4 (625) fuzzy rules. Fortunately, in most situations, all the rules do not contribute equally to the solution. Many methods are available to reduce the number of rules involved. One way would be to use a genetic algorithm optimization. Usually, however, a statistic-based processor can analyze the situation and give the contribution of each rule to the solution. Then the user can set the threshold for including rules at a level consistent with the specifications for precision and speed.

16.9 POTENTIAL IMPLEMENTATION ISSUES FOR EXPERT SYSTEMS

Potential problems in implementing expert systems in complex engineering systems can be projected from past experience with the introduction of new and innovative systems.

General Implementation Issues

1. Most complex engineering systems, as presently built and operated, are considered by the operators to be safe enough. With the possible exception of the severe accidents (i.e., Chernobyl, Bopough. etc.),

expert systems are not perceived to be needed to provide additional safety functions.

2. Introduction and use of an expert system must not introduce a new operational or safety problem. A thorough analysis of what could go wrong and what effect it could have on the plant and its safety system would be essential before implementation. The ultimate criterion in judging any new system is whether its failure can, in any way, lead to a challenge of existing plant protection systems.

Implementation Issues That Need To Be Addressed

A number of issues regarding the implementation of expert systems in complex engineering systems need to be addressed. These include, but are not limited to, the following:

1. *Quantitative and Objective Performance Guidelines for Expert Systems.* The primary concern about the introduction of any new system into a complex engineering system appear to be the impact it can have on the safety system when something goes wrong. The ultimate question in judging any new system must be "Can the failure of the expert system lead to a challenge of the existing safety systems?" Above all, replacement of an existing system with an expert system must not introduce new unresolved issues (i.e., new unreviewed safety hazards).

Introduction of a new system must not lead to confusion of operators or other plant personnel. New tools may be needed to evaluate and measure the performance of expert systems and the impact of these systems on human performance. Objective criteria that are quantitative in nature are needed.

2. *Validation and Verification (V & V).* In conventional software programming, verification and validation have well-established meanings; verification is a determination that software has been developed in a formally correct manner in accordance with a specified software engineering methodology; validation means demonstrating that the completed program performs the functions in the requirements specification and is usable for the intended purposes. However, expert systems go beyond the procedures of conventional software engineering, and a modularized, top-down, hierarchically decomposed design that makes conventional V & V possible may not be achievable. Expert systems, especially those operating under uncertainty or with incomplete data, may have so many states as to make exhaustive testing unfeasible. Hence, new approaches to V & V are needed for expert systems.

The inference engine may be considered simply as another digital computer program, and its V & V can be dealt with in the same way as with other digital computer programs (e.g., IEEE-5.3.2.1). The real problem is the adequacy of the knowledge base—that is, the qualifications of the expert whose expertise is incorporated into the knowledge base, the method used for acquisition of this expertise, and the method used to represent this expertise in the knowledge base. Except for relatively simple expert systems, exhaustive testing of the expert system or the knowledge base to cover all likely situations may not be adequate or feasible.

Generally, as a matter of policy, V & V should always be carried out by a group completely independent of the group that developed the expert system. Because V & V in expert systems is so intimately related to the design, true independence may extremely difficult to achieve. To the extent possible, the independence of the group that does V & V should be ensured by quality assurance procedures and organization policy.

3. *Human Factors.* A primary human factors concern is that the expert system should present information to the user in a way that is comprehensible and understandable. Information must mesh well with the perspectives used by the human, and the way in which the information is displayed should correspond to the user's mental model of the plant. The user should be able to understand the expert system's behavior.

Another concern is user reaction to the expert system. Will they like the system and accept it? Will they be comfortable with an expert system and use it when needed? Will they believe that the system will work and that it is useful? Above all, will they trust and have confidence in the information presented by the expert system? On the other hand, the user could become too dependent upon the guidance of an expert system and ignore other indications that might not agree with the conclusion of an expert system.

The function allocation and division of responsibility between the expert system and the human is another important issue. Humans should be assigned only those functions that they are most capable of performing and that utilize their abilities. Expert systems should relieve some of the physical and cognitive workload on users to avoid overload of the operators. The system should make human jobs more efficient. The expert system should be integrated with the other hardware, software, and tools in the user's work environment. Clearly, users should be involved in the design and analysis of the expert system and its interface with users.

REFERENCES

Bernard J., and Washio, T., *Expert Systems in the Nuclear Power Reactors*, ANS Publishing, La Grange Park, IL, 1989.

Coates, J., Artificial Intelligence: Observations on Applications and Control, *Computer Security Journal*, Vol. 5, No. 1, 1988.

Cooper, T. A., and Wogrin, N., *Rule-based Programming with OPS5*, Morgan Kaufmann, San Mateo, CA, 1988.

Feigenbaum, E. P., McCorduck, P., and Nii, H. P., *The Rise of the Expert Company*, Times Books, New York, 1988.

Gonzalez, A. J., and Dankel, D. D., *The Engineering of Knowledge-Based Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

Hertz, D. B., Boeing Has High Hopes for AI, *AI Week*, University of Miami, Coral Gables, FL, July 1, 1988.

Kaplan, S., Frank, M. S., Bley, D. C., and Lindsay, D. G., Outline of COPILOT, Expert System for Reactor Operational Assistance Using a Bayesian Diagnostic Module, *Proceedings of the International Post SMiRT-9 Seminar on Accident Sequence Modeling: Human Actions, System Response Intelligent Decision Support*, Munich, August 24–25, 1987.

Kruse, R., Schwecke, E., and Heinsohn, J., *Uncertainty and Vagueness in Knowledge Based Systems*, Springer-Verlag, Berlin, 1991.

Pin, F., Private communication, Oak Ridge, TN, 1992.

Ricker, M., An Evaluation of Expert System Development Tools, *Expert Systems*, Vol. 3, No. 3, 1986.

Sackett J. I., ed., *Proceedings of the ANS Topical Meeting on Artificial Intelligence and Other Innovative Computer Applications in the Nuclear Industry*, Snowbird, UT, August 31–September 2, 1987.

Swartout, W. , and Smoliar, S., On Making Expert Systems More Like Experts, *Expert Systems*, Vol. 4, No. 3, 1987.

Terano, T., Asai, K., and Sugeno, M., *Applied Fuzzy Systems*, Academic Press, Boston, 1994.

Uhrig, R. E., Applications of Artificial Intelligence in Nuclear Power Plants, *POWER Magazine*, McGraw-Hill Energy Systems, June 1988.

Van Horn, M., *Understanding Expert Systems*, Bantam Books, New York, 1986.

Waterman, D., How do Expert Systems Differ from Conventional Programs, *Expert Systems* Vol. 3, No. 1, 1986.

PROBLEMS

1. Discuss how expert systems can be verified and validated in the sense that software undergo verification and validation. Consider the different requirements for the inference engine and the knowledge base.

2. It is well known that the use of fuzzy rules has revived the use of expert systems. Explain what you believe to be responsible for this resurgence of interest of expert systems. Is the reason the same for all types of expert systems? If not, why?

3. Discuss the legal ramifications of using expert systems. If an expert system (or a neural network) fails in service and causes damages, who is responsible? The company selling the expert system? the user? The person who wrote the software for the expert system? The expert who supplied the information to the expert system? All of the above?

4. Discuss the relative advantages and disadvantages of having a deterministic model of a system in the knowledge base compared to having a neural network model in the knowledge base.

17

GENETIC ALGORITHMS

17.1 INTRODUCTION

Genetic algorithms as a field of study was initiated and developed in the early 1970s by John Holland (Holland, 1975, 1992) and his students, but its applications to real-world practical problems was almost two decades in developing. In one way or another, the primary purpose of using genetic algorithms is optimization. The specific nature of the problem or system to which optimization is being applied will determine the approach, the type of genetic algorithms used, and especially the evaluation or fitness function. There is no guarantee that a genetic algorithm will give an optimal solution or arrangement, only that the solution will be near-optimal in the light of the specific fitness function used in the evaluation of the many possible solutions generated.

In this chapter, the terms *chromosomes* and *genes* may appear to be used synonymously, but they are not. The meaning of these terms as used here is the same as that used by Goldberg (1989). Chromosomes are composed of genes which define the characteristics of the chromosomes and may take on several values called *alleles*. The position of a *gene* (its *locus*) is identified separately from the gene's function. Hence we can have a particular gene with a locus of position 12 whose allele value is brown. Generally, the *strings* of artificial genetic systems are analogous to chromosomes in biological systems and are often called chromosomes.

17.2 BASIC CONCEPTS OF GENETIC ALGORITHMS

Genetic algorithms mimic some of the processes of natural evolution. In doing so, some of the inherent features of evolution are utilized in fields far beyond genetics. However, there is no necessity that genetic algorithms as we use them mimic in detail the behavior of the evolutionary process. Indeed, users are free to utilize those features that are useful and discard aspects that seem unimportant in their applications. Since normal evolution processes are quite slow, biased reproduction, based on an aggressive "survival of the fittest" philosophy, is used to speed up the evaluation process.

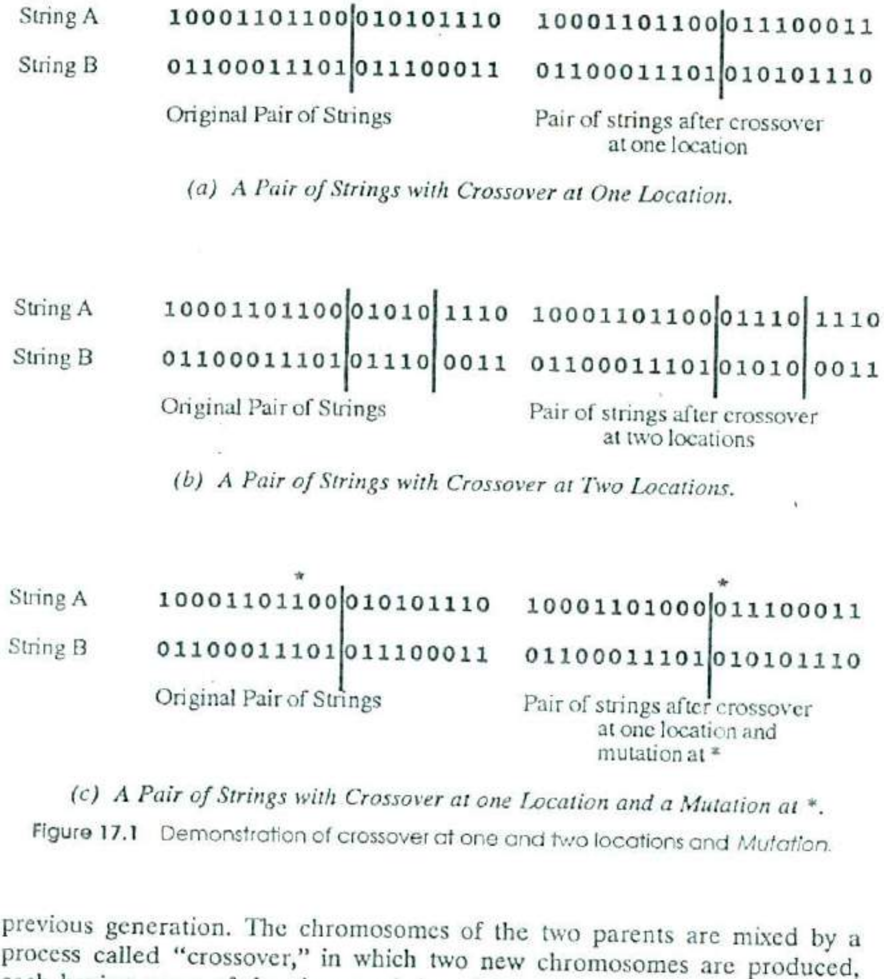
The mechanisms that induces evolution are not well understood, but the features of evolution have been investigated thoroughly. First, evolution takes place in chromosomes, the genetic units that encode the features and structure of living creatures. The specific descriptive features of a living creature is determined by the chromosomes of the previous generation, and evolution influences only these chromosomes, not the living creature from which they came. Since evolution is limited to chromosomes, living creatures do not evolve during their lifetime; their features, which are presumably set at the time of conception, are different from the previous generation only because the chromosomes of their parents changed through evolution.

Evolution, Natural Selection, and the Gene Pool

Natural selection is a process by which nature causes those chromosomes that encode better characteristics (by some criteria) to reproduce more often than those that encode poorer characteristics. Natural selection is the process that causes genetic algorithms to produce near-optimal solutions when the selected chromosome is decoded. This process involves creation of many chromosomes by *reproduction*, *mating* (*crossover*), *mutation*, and the survival of the chromosomes with the better characteristics. Successive generations of chromosomes improve in quality, provided that the criteria used for survival is appropriate. This process is often referred to as *Darwinian natural selection* or the *survival of the fittest*. In nature, this process of evolution occurs over many years, even hundreds or thousands of years. In the computer, the representations of chromosomes can undergo literally thousands of generations of change in a few seconds.

Historically, the characteristics of the chromosomes in genetic algorithms have been represented by 0s and 1s. All of Holland's work used this representation, and we will use it here. However, chromosomes can be represented by real numbers, permutations of elements, a list of rules, or other symbols. Binary representation is still the most common representation because the behavior of bit strings are more familiar and better understood.

Evolution takes place through the process of reproduction, which involves mutations and recombination after mixing of the parent's chromosomes to produce a creature that may have entirely different characteristics from the



(a) A Pair of Strings with Crossover at One Location.

(b) A Pair of Strings with Crossover at Two Locations.

*(c) A Pair of Strings with Crossover at one Location and a Mutation at *.*

Figure 17.1 Demonstration of crossover at one and two locations and *Mutation*.

previous generation. The chromosomes of the two parents are mixed by a process called "crossover," in which two new chromosomes are produced, each having some of the characteristics of the two parents. The two parent chromosomes split at some point, and one part of one parent chromosome is exchanged for the corresponding part of the other parent chromosome. The location of the crossover point at which the parents' chromosomes divide is apparently a uniform random process. If one of the new chromosomes obtains 75% of its characteristics from one parent and 25% from the other, the second new chromosome gets 25% form the first parent and 75% from the other. This process is illustrated schematically in Figure 17.1, where the chromosome is illustrated as a string of 0s and 1s.

Both of the resultant chromosomes go into the gene pool (where all the alleles reside), where they either replace poorer-quality chromosomes or are discarded. Once a chromosome is discarded, its unique features (good or bad) are lost forever. It is only through the processes of mutation (described below) that such a chromosome might possibly be recreated.

Mutation is a process by which a single component of a chromosome is changed randomly. It occurs in only a very small fraction (typically less than a fraction of a percent) of the chromosomes. It represents an abrupt change in the nature of the chromosome and influences all subsequent generations of chromosomes containing this mutated component. Of course, if this mutation results in a poorer-quality chromosome, it will be discarded and lost from the gene pool.

Each population has a gene pool consisting of a large number of chromosomes generated by the process of natural selection. The choice of which two chromosomes are mated and subject to the crossover process is somewhat random, but the fitter chromosomes are more likely to be selected first. New chromosomes are constantly being reproduced by the mating process described above, and those with better characteristics are retained while those with poorer characteristics are discarded. Generally, but not always, the mixing of chromosomes with quite different characteristics produce better chromosomes. As the process proceeds, the average quality of the gene pool improves because the poorer-quality chromosomes are discarded.

Objective Function–Fitness Function

The function on which an optimization algorithm operates—that is, seeking its maximum or minimum—is called the *objective function*. In neural networks, the objective function to be minimized is the mean square error over the entire training set. In genetic algorithms, the "fitness" is the quantity that determines the quality of a chromosome, from which a determination can be made as to whether it is better or worse than other chromosomes in the gene pool. The fitness is evaluated by a "fitness function" that must be established for each specific problem. This fitness function is chosen so that its maximum value is the desired value of the quantity to be optimized. Its importance cannot be overemphasized, because it is the only connection between the genetic algorithm and the problem in the real world. A fitness function must reward the desired behavior; otherwise the genetic algorithm may solve the wrong problem. Fitness functions should be informative and have regularities. However, they need not be low-dimensional, continuous, differentiable, or unimodal.

17.3 BINARY AND REAL-VALUE REPRESENTATIONS OF CHROMOSOMES

Weight representation in the chromosome has used both binary and real-value encodings, with binary being the more prevalent method. Binary coding of the weights can be implemented using either an ordinary binary representation of real-value weights or a corresponding Gray-scale binary encoding.

Table 17.1 Comparison of hamming distances in binary and gray coding

Decimal Number	*Binary Coding*	*Hamming Distance*	*Gray Coding*	*Hamming Distance*
0	0000		0000	
1	0001	1	0001	1
2	0010	2	0011	1
3	0011	1	0010	1
4	0100	3	0110	1
5	0101	1	0111	1
6	0110	2	0101	1
7	0111	1	0100	1
8	1000	4	1100	1
9	1001	1	1101	1
10	1010	2	1111	1
11	1011	1	1110	1
12	1100	3	1010	1
13	1101	1	1011	1
14	1110	2	1001	1
15	1111	1	1000	1
16	10000	5	11000	1

Binary and Gray-Scale[1] Representations

Gray scaling has the characteristic that the Hamming distance (the number of binary digits that change between successive decimal numbers) is always 1 compared to binary coding where the hamming distance is 4 in a four-bit representation (i.e., all bits change) as a decimal number changes from 7 to 8 (see Table 17.1). Such so-called Hamming cliffs can make genetic algorithms less stable. The reason is that change of a single digit due to mutation will usually cause a smaller change if the Hamming distance is small. In the binary code, half the changes have a Hamming value of 2 or more, whereas all changes in the gray scale have a Hamming distance of 1. Empirical studies (Caruana and Schaffer, 1988) on algorithms indicate that gray coding improves the process for some functions and performs no worse than binary coding in all cases.

There are several gray codings for any number, but the most commonly used Gray coding is the binary-reflected gray code. One simple scheme for

[1] Gray encoding or gray scale refers to the use of a binary code developed by F. Gray (1953), U. S. Patent #2-632-058 issued March 17, 1953.

generating such a gray code sequence is "start with all bits set equal to zero and then successively flip the rightmost bit that produces a new string." Table 17.1 compares binary coding with Gray coding for decimal numbers from 0 to 16. (At 16, the binary and gray codes must go to 5 digits since all possible combinations of 4 digits have been exhausted.) Interestingly, the 4-digit combinations used for representation of numbers from 0 to 15 are exactly the same for the binary and gray encodings, except that a specific combination of 4 digits represent different decimal numbers in the two codes. For instance, 0111 represents 7 in the binary code and 5 in the gray code.

Real-Valued Representations of Chromosomes

In real-valued encodings, the network weights are encoded as lists of real-valued weights. Crossover occurs across whole weights instead of occurring across bit strings representing weights. In mutations, incremental changes (plus or minus) are introduced into the real values. Davis (1991) discusses the advantages and disadvantages of real-valued encodings and argues that such encodings can yield superior results. Perhaps the main disadvantages of real-valued encodings are that robust parameters are not known and that specialized genetic algorithms may need to be tailored for each problem. Both of these disadvantages should be lessened as we gain more experience with genetic algorithms using real-valued representations of chromosomes.

17.4 IMPLEMENTATION OF GENETIC ALGORITHM OPTIMIZATION

Living creatures are probably the most complex systems in the universe. Hence, if evolution that involves reproduction with crossover, mutation, and natural selection can result in improvement of the species, it seems reasonable that the process will work to optimize other complex systems. Indeed, this has been the case, and the range of applications where genetic algorithms can optimize a process or system is limited only by the ingenuity of the user. Applications have now reached the point where many users are no longer versed in the details of how genetic algorithms operate; rather, they are concerned only with how the powerful capability of genetic algorithms, as presented in commercial software, can be utilized to optimize their particular problem or system.

Genetic algorithms do not rely on any analytical properties of the function to be optimized (such as the existence of a derivative). They are well suited to a wide class of problems, including optimization over parameter sets as well as global optimization of functions. However, before the genetic algorithm process can be carried out, two steps are necessary: (1) encode the variable to be optimized into a string of binary bits (or other appropriate representations) and (2) create an appropriate fitness function.

Bit-String Representation

It is necessary to structure bit strings to represent practical problems before undertaking a search for optimal conditions. Individual bit strings are organized to form an initial population of chromosomes. They can be generated randomly, but its is advantageous if the initial population of chromosomes can be somewhat related to the nature of the system being optimized. Genetic algorithms guide the string population to propagate from generation to generation to improve the survival probability of the entire population.

There are two approaches to determining which chromosomes to delete after a cycle of reproduction, crossover, and mutation. These are (a) the *generational approach*, where the entire population is replaced after each cycle, and (b) the *steady state approach*, where the members of both the old and the new gene pools with the highest fitness factor are retained. The generational approach tends to speed convergence, perhaps at the expense of diversity in the gene pool. The steady-state approach tends to produce somewhat better performance by retaining the best-performing bit strings, but the best solution may be missed because new genes that are the precursors of high-performing genes may be eliminated prematurely.

The convergence criteria for stopping the genetic algorithm is somewhat arbitrary. Generally, genetic algorithms converge rapidly, typically in a few hundred cycles or less. Stability in the value of the average fitness function from one generation to the next is generally the most appropriate criterion.

A related issue is the reproduction process where there are several options for selecting which genes should be reproduced. The two most common methods are *proportional selection* and *rank-based selection*. In proportional selection (discussed below) the number of times the gene can be reproduced is proportional to its fitness function. This technique, which was used by Holland, involves selecting the top performers and allowing multiple reproductions of the best performers. A sampling algorithm is usually used to allocate the number of reproductions to the various genes. The proportional method sometimes tends to give undue emphasis to superior performing chromosomes whose fitness functions may be 10 times the average fitness function. If such a super chromosome is reproduced 10 times in a pool of 50 genes, it would clearly distort the gene pool. In the rank-based selection process, each gene is typically reproduced only once, although there are variations of this algorithm that allow multiple reproduction of a single gene. Rank-based selection tends to converge slowly with less premature convergence and better diversity of the gene pool.

Reproduction

In the implementation of genetic algorithms, the reproduction process consists in the copying of individual strings according to the priority established by their objective function or fitness function f. (We will use the latter term in this chapter.) Copying strings according to their fitness function values means that candidates with higher fitness values have a greater probability of contributing one or more offsprings in the next generation. This is the "proportional" selection method discussed above. The selection probability for an individual string i (the ith string in the population) may be defined as

$$p_i = \frac{f_i}{\sum f_i} \tag{17.4-1}$$

where f_i is the fitness value of the ith individual in the population N.

The mating pool of the next generation is selected according to the probability p_i. Once an individual has been selected for reproduction, it is then entered in the mating pool, for further genetic operation action. If there is no overlapping between populations (i.e., the population size remains constant when a new generation replaces the old or parent generation), the expected number of reproductions of ith individual string is

$$n_i = N \cdot p_i = N \cdot \frac{f_i}{\sum f_i} = \frac{f_i}{\sum\left(\frac{f_i}{N}\right)} = \frac{f_i}{\bar{f}} \tag{17.4-2}$$

where $\bar{f}$ is the average fitness of the population. This agrees with our earlier thesis that the best chains are more likely to be reproduced.

17.5 FITNESS FUNCTIONS

The fitness function of a genetic algorithm can be designed to perform different search tasks of optimization. The value of fitness function is the quantity to guide the reproduction process in the genetic algorithms for creating the next generation. Usually, the fitness function is designed in a way that its values are all positive, and the higher the value of the fitness function, the better the performance of the individual bit string in the population. A higher value of the fitness function also means that the individual bit string gets the better chance to be selected for production of the next generation. These guidelines indicate that the fitness function is a function of the number of inputs selected (the fewer the number, the greater the fitness) and the network training error (the smaller the value, the greater the fitness).

To illustrate the role of the fitness function and the general process involved in using genetic algorithms for optimization, two examples are provided. In the first, Example 17.1, a quadratic function $y(x) = 1 - x/10 + x^2/200$ is to be optimized for its maximum or minimum value. In Example 17.2, an actual problem is used to illustrate the selection of the fitness function.

Example 17.1 Simple Hand-Calculated Example of the Genetic Algorithm Process. Let us assume that we want to optimize (i.e., find the minimum or maximum value of a function)

$$y(x) = 1 - \frac{1}{10}x + \frac{1}{200}x^2 \tag{E17.1-1}$$

using a genetic algorithm process. Of course, we can readily determine that this function has a minimal value of 0.5 at $x = 10$ by other means. Please understand that this problem has been contrived to demonstrate the methodology of genetic algorithms. Because of the small size and small number of binary strings, the behavior of the process is not representative of that in real world genetic algorithms.

We can create the initial population by flipping a coin to select our mating pool, which in this simple example consists of five 5-bit random binary sequences. These are listed in Table 17.2 as "String x," and their binary values converted to the base 10 are listed as "Value x." The function $y_i(x)$ is then evaluated using the above formula. Clearly, $y(x)$ is related to the fitness function since it represents the quantity we want to optimize. In this case, the optimal value is a minimum, and the genetic algorithm process gives the maximum value of the fitness function. Hence, it seems reasonable to let the fitness function be the reciprocal of $y(x)$; that is, $f_i(x) = 1/y_i(x)$. Generally, however, we do not have a formula of the quantity to be optimized, and the fitness function has to be selected on the basis of data available and the nature of the problem involved (see Example 17.2). Note that the last two columns in Table 17.2 are the selection probability for an individual string and the expected number of reproductions of the ith individual string as given by equations (17.4-1) and (17.4-2) respectively.

Table 17.2 Hand calculations for a genetic algorithm

	String	Value	$y_i(x)$	$f_i(x)$	$f_i/\sum f_i$	$f_i/\bar{f}$
	x	x				
v_1	10111	23	1.345	0.743	0.101	0.504
v_2	01100	12	0.520	1.923	0.261	1.304
v_3	10100	20	1.000	1.000	0.136	0.677
v_4	00110	6	0.580	1.732	0.235	1.173
v_5	01001	9	0.505	1.980	0.267	1.342
Sum				7.378	1.000	5.000
Avg.				1.476	0.200	1.000
Max				1.980	0.267	1.342

Now, let us carry out *reproduction*, *crossover*, and *mutation* operations on string x. This is shown in Table 17.3. The strings are randomly mated with other strings at the indicated crossover points to produce new strings. Furthermore, mutation takes place in the middle digit of value v_1 (indicated by an asterisk in Table 17.3), where a 1 changes to a 0 after crossover has taken place.

This mutation changes the fitness function of v_1 from a 1.000 to a 1.471, a 47.1% increase. Table 17.3 shows the results of reproduction, crossover, and mutation on the group of five chromosome strings. Crossover between each mated pair of genes in the pool produces two new chromosomes which are given in the column headed "new population." These new genes are evaluated to give their fitness functions listed under the column labeled $f_i(x)$. The double asterisk indicates the five genes with the highest fitness functions that are to be reproduced in the next generation if rank-based selection is used. Of these five genes, three have much higher values of $f_i(x)$. Hence, if proportional selection is used, the two genes with lower values of $f_i(x)$ would probably be replaced with duplicates of the two genes with the highest values of $f_i(x)$.

It is interesting to note that the average fitness function of the original five strings is 1.476 (see Table 17.2) compared to an average fitness function of 1.459 (see Table 17.3), a 1.1% decrease for the 10 new chromosomes produced by crossover and mutation. However, the fitness function of the five new chromosomes selected for reproduction is 1.821, an increase of 23% over the original five chromosomes. This increase is unusually high for one cycle due to the small length of the strings, which tends to increase the impact of even a change in a single digit.

An examination of the new set of strings indicates that new v_5 could be a "super" string that could dominate future cycles of reproduction. This is not desirable, particularly in the early part of the optimization process, because it could lead to a premature selection of an optimum which was not a true optimum. The concern here is prematurily limiting the gene pool which could cause the process to select a local minimum or maximum rather than a global value. There are a number of techniques available to avoid this problem which the reader can find in literature that specializes in genetic algorithms (Holland, 1975, 1992: Goldberg, 1989; Davis, 1991).

The generation, crossover, and mutation processes continue until there is no significant change in the average value of the fitness functions. At that point, it is necessary to decode the string to identify the optimal value, a minimum in the function in this case. Let us use the largest value of fitness function (2.000) to represent the optimal case. Because of the reciprocal relationship, the optimal value of $y(x)$ is 0.500. If we substitute this value into the equation for y, we get

$$y(x) = 1 - \frac{1}{10}x + \frac{1}{200}x^2 = 0.5 \qquad \text{(E17.1-2)}$$

Table 17.3 Reproduction, crossover, and mutation on Strings v_i

	MATING POOL (REPRO-DUCTION)	MATE RANDOMLY SELECTED	CROSSOVER SITE RANDOMLY SELECTED	NEW POPULATION	VALUES			
					x_i	$y_i(x)$	$f_i(x)$	$f_i(x)$ **
				*				
v_1	101 | 11	2	3	10000	16	0.680	1.471	
				01111	15	0.625	1.600**	1.600
v_2	01 | 100	5	2	01001	9	0.505	1.980**	1.980
				01100	12	0.520	1.923**	1.923
v_3	1010 | 0	1	4	10101	21	1.105	0.905	
				10110	22	1.220	0.820	
v_4	0 | 0110	3	1	00100	4	0.680	1.471	
				10110	22	1.220	0.820	
v_5	010 | 01	4	3	01010	10	0.500	2.000**	2.000
				00101	5	0.625	1.600**	1.600
						Sum	14.589	9.103
						Avg	1.459	1.821
						Max	2.000	2.000
				* MUTATION			**GENES SELECTED FOR NEW GENE POOL	

INFLUENCE OF MUTATION UPON CHARACTERISTICS OF v_1

		x_i	$y_i(x)$	$f_i(x)$
BEFORE MUTATION	10100	20	1.000	1.000
AFTER MUTATION	10000	16	0.680	1.471

We can then solve the quadratic equation (E17.1-2) for the value of $x = 10$ as the location of the optimal value, which is then calculated to be 0.5.

Clearly, this example was contrived to give good results with only a single cycle. The short length of the bit string and the small size of the gene pool accentuate the effect of the processes involved. In a practical problem, the typical gene pool may have 50 to 200 chromosomes and go through hundreds of cycles. However, pools of over 50,000 chromosomes and tens of thousands of cycles have been used. A large population gives more diversity and better final solutions, but longer computational times are involved. Pools of less than 30 chromosomes are subject to premature convergence because stochastic effects tend to dominate the behavior of the genetic algorithm. □

Example 17.2 Evolution of a Fitness Function.[2] In this example, a large neural network with 25 inputs (which are instantaneous values of 25 different measured parameters) and 8 outputs (representing 7 different transients and a no transient state) is used to diagnose transients in a nuclear power plant. Every half-second, the 25 measured values are applied to the neural network whose output indicates almost instantaneously which transient is occurring or that there is no transient. The neural network is trained on transients generated in a full scope, high fidelity nuclear power plant simulator. A complex recurrent backpropagation neural network was needed to model the plant dynamic behavior because of the complex interrelations between the variables.

A sensitivity analysis as described in Section 8.5 was used to determine the most important inputs (typically 4 to 6 inputs) needed for the detection of each specific transient. This allowed the use of "modular" neural networks,[3] small backpropagation networks without recurrent connections with only a few inputs and a single output for each transient. Subsequent tests indicated that the modular neural networks were equally as effective in detecting transients as the large master neural network with 25 inputs and 8 outputs. The problem was that the master neural network had to be created before the sensitivity analysis could be used to determine the most important inputs for the "modular" neural networks.

Genetic algorithms were selected as an alternate method of determining the most important (optimal) variables for the modular networks without having to create and train the master neural network. The fitness function needs to be defined to guide the search for the best combination of inputs for the individual modular networks. The fitness function may have different forms for different optimal search tasks. It needs to be defined to guide the

[2] This example was developed by Zhichao Guo in his Ph.D. dissertation in Nuclear Engineering entitled "Nuclear Power Plant Diagnostics and Thermal Performance Studies using Neural Networks and Genetic Algorithms," University of Tennessee Library, Knoxville, TN, 1992.

[3] The term "modular" as used here does not refer to modular neural networks as described in Section 8.8.

search for the best combination of inputs for different modular networks. The criterion chosen was that the fitness function should be designed to (a) select fewer variables while making the training error smaller for a fixed number of training cycles and (b) penalize the selection which chooses more than necessary inputs and/or makes training error larger. This can be expressed in equation form as

$$\text{fitness} = \frac{c_1 \cdot (\text{total number of inputs})}{\text{number of inputs selected}} + \frac{c_2}{\text{network training error}} \qquad \text{(E17.2-1)}$$

where c_1 and c_2 are constants based on the conditions of the problem. Tests carried out using the fitness function of equation (E17.2-1) failed because the resultant neural network training error was too large to be accepted. The fitness function was the sum of two terms which was dominated by the first term associated with a small number of inputs without regard for the influence of the training error.

This experience led to the proposal of a second trial fitness function of the form

$$\text{fitness} = (1 \;\text{[illegible]}\; -1)^{1.5}) \cdot e^{-0.01 y^{1.5}} \qquad \text{(E17.2-2)}$$

where

$$x = \frac{\text{total number of inputs}}{\text{total number of inputs selected}}$$

and

$$y = \text{network training error (\%)}$$

Equation (E17.2-2) is the product of the two exponential terms. The first term reflects the rewarding and penalizing for the number of inputs. Its value is exponentially decreasing with increasing number of inputs. The second term reflects the rewarding for a small training error and penalizing of large errors. The numerical coefficients were selected after experimentation involving the expected minimum number of inputs and expected smallest value of error. Tests indicated that this fitness function gave improved results but still had difficulties. For instance, reducing training error continuously improved the fitness function until an error of about 4% was reached, after which further reductions did not change the fitness function. Furthermore, strings which exhibited large fitness values in the early stages tended to dominate the next generation gene pool. In the case of the fitness function of equation (E17.2-2), one specific string produced 14 of the 20 strings in the pool.

The experience with the second fitness function led to the proposal of a third fitness function of the form

$$\text{fitness} = \left(1 - e^{-(x-1)^{0.15(x+1)}}\right) \cdot e^{-0.01 y^{0.7(x+1)^{1/3}}} \qquad \text{(E17.2-3)}$$

where, x and y are defined as in the previous fitness function. This fitness function was found to provide the appropriate influence of the number of inputs and the training error without the undue influence of early strings having-large fitness functions.

Subsequent comparison of the input variables selected for the "modular" neural networks by sensitivity analysis and by genetic algorithms showed that the two most important inputs for each of the seven transients and the normal states were almost always the same. Beyond the second most important variable, there were a number of inconsistencies. However, tests showed that the training errors after a prescribed number of cycles for the networks selected by the two methods were substantially the same and that the networks performed equally well.

It is seen that this fitness function was arrived at by a series of "trials and errors." Clearly, experience in working with fitness functions gives insight into the form of the fitness function. However, if specific information that is useful in forming the fitness function is available, it should be used. □

17.6 APPLICATION OF GENETIC ALGORITHMS TO NEURAL NETWORKS

A number of researchers have tried to connect the genetic algorithms with neural networks in recent years. Whitely and co-workers (Whitely and Bogart, 1989, 1990; Whitely and Starkwerther, 1990) used genetic algorithms to guide a backpropagation based neural network in finding the necessary connections instead of full connections in the GENITOR II software in order to enhance the speed of training. They also used this software to optimize small networks with the result that the resultant networks learned much faster and much more consistently than fully connected networks. Koza (1990) used genetic algorithms to guide search for the time-optimal "bang-bang" control strategy for the cart-centering problem, a version of the broom balancing problem, with the additional constraint that the cart be located at a specific location during operation, by genetically breeding populations on control strategy. Maricic and Nikolov (1990) used the neural network designer GENNET to find the most appropriate network architecture for solving a given problem. In GENNET, a genetic algorithm block is responsible for generating population architectures, which are used to create a set of stand-alone backpropagation networks. Interactions between genetic algorithms and neural networks generate the best network architecture. Garis

(1990) used a genetic algorithm to train modular neural networks by finding the proper weights for the full connections. Genetic algorithms have been used to guide the design of neural control circuits, which combine the neural modules to form functional hierarchies. Muselli and Ridella (1990) proposed a combination method of genetic algorithms and simulated annealing to generate and choose the set of points in the network connection weight space to speed up reliability and convergence.

Combining Neural Networks and Genetic Algorithms[4]

Genetic algorithms typically encode the parameters of artificial neural networks as a string or list of the network's properties. The algorithm requires that there be a large population of these lists or strings (chromosomes) representing many possible parameter sets for the given network. The utilization of genetic algorithms for optimization lends itself easily to parallel computers. Advantages of using parallel techniques include the ability to search the entire weight space versus localized search in the weight space via a gradient descent technique. This global aspect of the search helps avoid local minima which can occur with other gradient descent techniques. Combined genetic algorithm–neural network technology (sometimes called GANN) have the ability to locate the neighborhood of an optimal solution quicker than backpropagation methods due to its global search strategy, but once in the neighborhood of the optimal solution, the GANN algorithm tends to converge to the optimal solution slower than backpropagation methods. This is because the final convergence of the genetic algorithm from the optimal neighborhood to the optimal solution is controlled mainly by the mutation operators. Drawbacks of the GANN technology are the large amount of memory required to maintain a viable population of chromosomes for a given network and some question as to whether this technique scales to larger network sizes.

The most common implementations of neural networks and genetic algorithms use direct encoding strategies that directly encode network parameters, such as weight values and network connectivity to optimize the weights and/or architecture of a given artificial neural network. When optimization is confined to the weights of a given neural network (i.e., the structure is fixed), the network weights are encoded as genetic strings (chromosomes) or lists of parameters. A large population of these strings, where each string represents an instance of a network's parameters, is then combined using the genetic operators of crossover and mutation to form the next generation of chromosomes based on their fitness function. These fitness functions are often taken as the inverse of the network error (yielding a large number for good weights) scaled by the sum total error of the population.

[4] Part of this section was taken from a class report prepared by James R. Cain, a graduate student at the University of Tennessee in 1995–1996.

Using genetic algorithms to optimize the architecture of an artificial neural network is carried out in a similar fashion with the network connectivity of the neurons being encoded into the chromosomes. Because of the large and initially diverse population, a larger area of the weight space is much more likely to be searched compared to more traditional gradient descent techniques.

A combination of neural network and genetic algorithm training methods (called the Lamarkian learning method) involves periods of genetic optimization in between periods of backpropagation training. This method provides a powerful method for combining gradient descent techniques (like backpropagation) with evolutionary optimization techniques encompassed in the genetic algorithms.

An alternative approach is the Baldwin learning method which utilizes the backpropagation algorithm to adjust the fitness value for chromosomes. Hence, chromosomes that show the ability to learn through the backpropagation algorithm are considered to be more fit and therefore more likely to be selected to pass their genetic material onto subsequent generations.

Most common coding strategies employ connection-based systems which allow for weight and connectivity optimization of a predefined architecture. Issues which must be addressed before the start of training include population size, binary versus real valued weight representation, how many bits to use if binary chromosomes are used, type of crossover used, the prevalence of mutation, whether to use rank-based roulette wheel or proportional selection, and the criterion for stopping the process.

17.7 FUZZY GENETIC MODELING

As discussed earlier, fuzzy systems are made up of fuzzy sets, defined by their membership functions and fuzzy rules that determine the action of the fuzzy systems. Fuzzy systems can model general nonlinear mappings in a manner similar to feedforward neural networks since it is a well-defined function mapping of real-valued inputs to real-valued outputs. Kosko (1992) has shown that fuzzy systems, like feedforward neural networks, are universal approximators in that they are capable of approximating general nonlinear functions to any desired degree of accuracy. All that is needed for practical application is a means of adjusting the system parameters so that the system output matches the training data. Genetic algorithms can provide such a means. Furthermore, fuzzy systems are effectively transparent in that everything that happens is clearly apparent. Each fuzzy associate memory (FAM) matrix entry is just a fuzzy rule that is easy to understand. This is very different from neural networks where the weight matrix, the most visible parameter, is virtually uninterpretable. Furthermore, a fuzzy system has the capability to analyze the distribution of training data versus the distribution of test data. If these are radically different, then one knows in advance that the results of the mapping will not be satisfactory.

Fuzzy rules can be concisely represented with one or more FAM matrices. In some cases, the FAM matrix can be established on the basis of a person's knowledge of the system. If such information is not available, then genetic algorithms can be used to establish the FAM matrix.

Optimizing a FAM Matrix Using Genetic Algorithms

Earlier the FAM matrix in fuzzy systems was discussed as an alternative to a neural network to relate or model complex inputs and outputs when they were represented by fuzzy sets in a fuzzy variable. This arrangement was particularly attractive when there were two inputs, and the overall behavior was intuitively obvious or the relationship could be derived from simple experiments. When this was not the case, the FAM matrix has to be trained from data available in ways similar to the training of neural networks. This section discusses the use of genetic algorithms to optimize the training of the FAM matrix.

A fuzzy system has a number of parameters that define fuzzy sets that are candidates for optimization. While optimization of several variables simultaneously is possible, it is much simpler and more practical to optimize only one variable at a time. This is usually possible if the general nature of most variables are known or at least bounded.

A fuzzy system has several parameters that can be optimized using genetic algorithms. Included are fuzzy sets used for input and output variables, the membership functions that define fuzzy sets, the structure and entries in the FAM matrix, and, in some cases, the weight assigned to each rule. Welsted (1994) presents such an example where the FAM matrix entries are optimized because they have the most influence in determining system output. In that example, adaptation is accomplished through the minimization of an error function. The approach used by Welsted is to convert the matrix entries into a long binary string. Since each matrix entry is a string of 1s and 0s, the linking together end to end of these entries creates a very long binary vector. This is the chromosome used in the optimization in genetic algorithms.

Welsted's example problem (interest rate modeling) is structured to use a single FAM matrix that deals with all five inputs simultaneously. However, with five input variables, the FAM matrix is a five-dimensional hypercube. With three fuzzy sets per variable (negative, zero, and positive represented by a "left shoulder," a trapezoid, and a "right shoulder," respectively), the FAM matrix has 3^5, or 243, entries. This is the "curse of dimensionality" referred to by Kosko (1992). The number of fuzzy sets per input determines how finely we look at a problem and how much data we have to have for training. Each FAM matrix is an output fuzzy set represented by a three-bit representation; hence there are eight (2^3) output fuzzy sets.

If only one item in each matrix entry is activated, we need 8 × 243 or 1944 items of information just for training. This problem is equally serious in neural networks where increasing the number of inputs increases the number of data sets needed for training to cover the dynamic range over which the variables may change. Inadequate data in either fuzzy systems or neural networks will result in regions of the state space not being covered.

The training (adapting) and testing phases of this process are similar to the training phase of neural networks. The system is "initialized" by setting initial input values and corresponding initial output values. The quantity to be minimized is the error accumulated over the training set between the fuzzy system output and the desired output. Since the fitness function is to maximized, it is defined as a constant minus this accumulated error. Training is accomplished by running the genetic algorithm operating on the fuzzy system fitness function. The outputs of the genetic algorithm training process are the coordinates of the defining values of the fuzzy sets. Every time a new optimal value is attained, the fuzzy system is saved to file.

Since the genetic algorithm is used only to minimize the error in the training process, it is not used after training is complete. Use of the FAM matrix to relate inputs and outputs proceeds in a normal manner.

17.8 USE OF GENETIC ALGORITHMS IN THE DESIGN OF NEURAL NETWORKS

At the present time, there is no generally accepted theory or methodology for the design of neural networks, and the process used is generally a trial-and-error approach based on the experience of the designer. The complexity of neural network design arises from the high-dimensional, heterogeneous space that must be explored by the system. The primary features, that are of concern in the design of neural networks are the structure of the network, the inputs to the networks, and the specification of the learning algorithm parameters. All of these quantities are problem-specific. While there are guidelines based on experience that can be very helpful in the design, some mathematical-based procedure would be very helpful. Optimization of the design based on the use of genetic algorithms offers such a methodology.

Use of Genetic Algorithms in Selecting Neural Network Structure

The Honeywell Technology Center (Harp and Samad, 1994) has developed an approach for designing and utilizing genetic algorithms for optimizing neural networks for use in modeling of complex systems. Their experience shows that the simultaneous optimization of network inputs, structure, and learning parameters is crucial for accurate modeling.

Usually, all of a network's parameters are encoded as genes in a chromosome in the form of a string of bits. Genetic algorithms procedures are then used to manipulate these chromosomes to produce improved parameters represented by the bit strings. Honeywell personnel (Harp and Samad, 1994) extended the concept of chromosomes and bit strings to a tree-structured entity with three generic families of genes: bytes, sequences, and structures. Byte genes represent scalar-valued parameters (e.g., learning rates). Sequence genes are ordered collections of other genes in which all the genes within a sequence are of a given specified type. Structure genes are fixed length ordered collections of genes of given types, with the type being determined by position. In an analogy to trees, the leaves are byte genes while the branches are formed by sequence and structure genes. An individual genetic tree is a sequence of structure genes representing areas and related connectivity that correspond loosely to a layer of a neural network but are more broadly applicable. Each area structure gene parameterizes the area in terms of an address, its number of neurons, the connecting weights, the learning parameters, and so on.

Harp et al. (1989, 1990) used a "blueprint" scheme to manipulate genetic algorithm representations of how sets of neurons are connected. In this work, network characteristics are represented by a blueprint, defined as a data structure that encodes various characteristics of the network including structural properties, input selection, and learning algorithm parameter values. A blueprint is instantiated into an actual network, and the neural network is trained using a learning algorithm and the learning parameters specified in the blueprint. The trained neural network is then evaluated using testing data, including an evaluation of its robustness by disabling some neural units or perturbing the learned weight values. Then its fitness is computed. The fitness estimate can be an arbitrarily complex function, such as the weighted linear sum of relevant criteria such as the number of nodes and weights in the network, accuracy, learning speed, efficiency, average and maximum number of outgoing weights from a node, and the various test scores.

After the evaluation, the next generation of the network is formulated in the blueprint. This process is mediated by a number of genetic operators (crossover, mutation, etc.) in which two blueprints are spliced together to produce a child blueprint. In effect, the genetic operators are being applied to these blueprints on a macroscale whereas genetic algorithms apply these genetic operators to bit strings on a microscale. The advantage of using the overall approach described here compared to "manual optimization" is that it allows the developer of the neural networks to explore large amounts of design space that would otherwise be left unexplored.

REFERENCES

Caruana, R. A., and Schaffer, J. D., Representation and Hidden Bias: Gray vs. Binary Coding for Genetic algorithms, in *Proceedings of the Fifth International Congress on Machine Learning*, J. Laird, ed., Morgan Kaufmann, San Mateo, CA, 1988.

Davis, L., ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY, 1991.

Garis, H. de, Modular Neural Evolution for Darwin Machine, *Proceedings of IJCNN-90-Wash.-DC*, Washington, DC, January 1990.

Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

Gray, F., Pulse Code Communication, U. S. Patent #2-632-058, March 17, 1953.

Gruau, F., and Whitley , D., Adding Learning to the Cellular Development of Neural networks: Evolution and the Baldwin Effect, in *Evolutionary Computation*, Vol. 3, No. 1, MIT Press, Cambridge, MA, 1993, pp. 213–233.

Guo, Z., Nuclear Power Plant Diagnostics and Thermal Performance Studies Using Neural Networks and Genetic Algorithms, University of Tennessee Library, Knoxville, TN, 1992.

Harp, S., Samad, T., Genetic Optimization of Neural Network Architectures for Electric Utility Applications, Final Report, Electric Power Research Institute, Research Project No. 8016-04, Palo Alto, CA March 1994.

Harp, S., Samad, T., and Guha, A., Designing Application Specific Neural Networks Using the Genetic Algorithm, in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, ed., Morgan Kaufmann, San Mateo, CA, 1990.

Harp, S., Samad, T., and Guha, A., Towards the Genetic Synthesis of Neural Networks, in *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1989.

Holland J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

Holland J. H., Genetic Algorithms, *Scientific America*, Vol. 267, pp. 66–72, 1992.

Kosko, B., *Neural Networks and Fuzzy Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

Koza, J. R., and Keane, M. A., Cart Centering and Broom Balancing by Genetically Breeding Populations of Control Strategy Programs, in *Proceedings of IJCNN-90-Wash.-DC*, Washington, DC, January 1990.

Moricic, B., and Nikolov, Z., GENNET—System for Computer Aided Neural Network Design Using Genetic Algorithms, *Proceedings of IJCNN-90-Wash.-DC*, Washington, DC, January 1990.

Muselli, M., and Ridella, S., Supervised Learning Using a Genetic Algorithm, *Proceedings of INNC-90*, Paris, France, July 9–13, 1990.

Welsted, S. T., *Neural Network and Fuzzy Logic Applications in C/C++* , John Wiley & Sons, New York, 1994.

Whitely, D., and Bogart, C., Optimizing Neural Networks using Faster, More Accurate Genetic Search, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 1989.

Whitely, D.,and Bogart, C., The Evolution of Connectivity: Pruning Neural Networks using Genetic Algorithms, *Proceedings of IJCNN-90-Wash.-DC*, Washington, DC, January 1990.

Whitely, D., and Starkwerther, Optimizing Small Neural Networks Using a Distributed Genetic Algorithm, *Proceedings of IJCNN-90-Wash.-DC*, Washington, DC, January 1990.

PROBLEMS

1. Discuss the relative merits of the "proportional selection" and the "rank-based selection" of the surviving chromosomes. Which is easier to implement?

2. The "fitness function" is by far the most important quantity in the use of genetic algorithms. One method of selecting this function was illustrated in Section 17.5. Discuss other means of determining the most appropriate "fitness function." Can you envision a method of optimizing the selection of the "fitness function" used in an optimization process?

3. Consider the data presented in Table 10.1 for welding tests. If you wanted to use genetic algorithms to optimize the travel speed and arc current for a given configuration (thickness, bead width, and bead penetration) of a weld, how would you go about it? What kind of "fitness function" would you develop? How would you go about ensuring that the "fitness function" was appropriate?

4. Carry the reproduction, crossover, and mutation processes on for another cycle using the information provided in Table 17.3. Is there significant improvement (or deterioration) of the 'fitness" of the chromosomes? If so, why? If not, why?

5. An alternate fitness function for Example 17.1 is

$$f_i(x) = 1 - \frac{y_i(x)}{2.71}$$

The denominator 2.71 is $y_i(x)$ when $x = 31$, the largest possible value of x for a 5-bit binary string. Evaluate $f_i(x)$ and $f_i(x)^{**}$ in Table 17.3. Are the new genes selected the same? If so, why? If not, why not?

18

EPILOGUE

18.1 INTRODUCTION

In the late 1960s, the American Society for Engineering Education (ASEE) issued a special report entitled "Goals of Engineering Education" in which the authors looked into their crystal balls to the year 2000 and predicted the types of projects on which engineers would be working during the next one-third century. The point of the study was that the engineering students in school at the time of the study (1967–1968) would still be active in the engineering profession at the turn of the century, and it was the engineering educators' responsibility to see to it that educational experiences at engineering colleges constitute proper preparation to meet the challenges that would arise in the rest of the century. Among the projects they predicted were:

Large-scale ocean farming

Fabrication of synthetic protein

Controlled thermonuclear power (fusion energy)

Regional weather control

Correction of hereditary defects by molecular (genetic) engineering

Automated high-IQ machines (expert systems and artificial intelligence in general)

Universal language through automated communications

Mining and manufacturing on the moon

Directed energy (microwave and laser) beams

Commercial global ballistic transports (the "Tokyo Express")

With less than half a decade to go, many of these predictions for the year 2000 are well on their way toward reality, while others, though feasible, are not being given serious consideration today. Perhaps artificial intelligence in the more general form of "soft computing" has had as much, if not more, impact than any of the other technologies listed above. Yet, we have only seen the tip of the iceberg as far as its influence on the future.

As far out as some of the items listed by the ASEE seemed in 1967, most of them are accepted as legitimate areas for engineering involvement today. However, some of the things appearing on the horizon today virtually defy our imaginations. For example, it has recently been reported that a team of scientists at the Max Planck Institute has opened a two-way communication link between a silicon chip and a biological neuron, effectively establishing a signaling channel that works in both directions. (ACM, 1995). The chip stimulates a leech's nerve cell through induced charges, and while capable of communication, no electrical current flows between the neuron and the chip (an essential requirement for any prosthetic limb controlled by the brain through a living nervous system).

18.2 IS ARTIFICIAL INTELLIGENCE REALLY INTELLIGENT?

In her book entitled *In Our Own Image*; *Building an Artificial Person*, Maureen Caudill (Caudill, 1994) examines the current state of technology and the accelerating trend in developments of robots, computer vision, understanding speech, sensing, diagnostics, and so on, and concludes that the construction of an "artificial person" is closer than most of us believe. Clearly, the first such "artificial person" would not be a Commander Data of Star Trek fame, but the essential processes to sustain "artificial life" are perceived by Caudill to be feasible in the twenty-first century. While we take no position on this well-documented but controversial thesis, we will point out that most of the advances in the technologies listed under soft computing have their origin in biological processes of humans, especially physiological processes and psychological behavior of the brain.

Many scientists and engineers are somewhat skeptical about artificial intelligence, especially in the light of the "excessive claims" of some of the pioneers in expert systems and neural networks. Fuzzy systems avoided this pitfall only because it was quietly developed in Japan without much fanfare until successful systems were being demonstrated. The skepticism comes from deep and privately held gut feelings that computers can never "be like" or "live like" humans. They are perceived as "just machines," and hence by definition they cannot be intelligent. We often tend to use anthropomorphic terms like "intelligence" to describe their workings in the age-old tradition of projecting something of ourselves and nature to the artifacts we made. It may well be that computers are no more intelligent than locomotives are iron horses. Artificial intelligence may simply be an inspiring metaphor for pursuing the enhancement of human intelligence. Indeed, concepts that have sprung out of artificial intelligence are revolutionizing the workplace, the office, the school, the marketplace, and the laboratory. In the manufacturing sector, soft computing is changing not only the design and analysis, but also the physical manufacturing with àdded flexibility and benefits in scheduling, production, maintenance and managment. The most often heard terms in manufacturing these days are *agile manufacturing* and the *virtual company*, concepts that are considered to map directly. to advancements in *soft computing* and its implementations.

The fundamental characteristics of emerging new products and systems may be quite different from these we deal with today. Agile or flexible systems will, of necessity, be more *proactive*. Whether they are intelligent agents or subjective objects (like today's machines that respond only to present conditions), is less important than the fact that they get the job done for you. Already a trend is underway in Japan toward predictive and anticipatory systems using predictive fuzzy control. Examples include control systems in many Japanese elevators, the Sendai metroliner, and the *Fugen* nuclear power plant.

In the future, technology that is user-friendly to people and capable of self-adjustment to custom fit individual needs will become important. In order to bring this flexibility, we see neurofuzzy technologies becoming part of the man-machine interface. They truly hold considerable promise to harmonize and enhance the relation between humans and machines and make it possible to incorporate actions, judgments, and thoughts that are near-human into a wide variety of devices and systems. Neurofuzzy technologies that respect the users' subjective desires, backgrounds, and idiosyncracies are expected to find their way into a great variety of products, making it possible for machines to say things like "Is this what you are trying to say?" or "Is this what you really want?" In industry, this may be particularly suitable for addressing bad structure problems for which computerized control has until now been difficult; it may indeed make automatic operation equivalent to operation by skilled operators.

18.3 THE ROLE OF NEUROFUZZY TECHNOLOGY

The principal topic in this book that is not commonly covered in other books or in university courses in the science and engineering fields is the *neurofuzzy* methodologies of Chapters 12 through 15. In Chapter 12, artificial neurons that utilize fuzzy operations (e.g., max, min, etc.) in place of multiplication and addition, as well as neural networks that also utilize fuzzy processes, are described. Applications such as "Fuzzy ARTMAP"and "fuzzy clustering" are already widely known in the artificial intelligence community and beginning to be utilized in engineering research and development.

In Chapter 13, we introduced neural methods into fuzzy systems. The overriding issue in fuzzy systems, whether they be used for expert systems, decision-making, or control, is defining the linguistic *if/then* relationships that constitute the algorithm on which the process is based. Neural networks and/or neural processes with their ability to extract information from examples (learning) can play an essential role in providing a better basis for fuzzy algorithms. Defining membership functions for fuzzy variables by using neural network is a very valuable process.

Chapter 14 is the result of a computerized literature search of scientific and technical journals for titles of articles that include both the words *fuzzy* and *neural*. Out of about 700 such publications, we chose about 50 examples from 12 fields where neurofuzzy systems were used advantageously. The purpose of this chapter was to illustrate the wide range of applications of neurofuzzy systems.

In Chapter 15, examples of research carried by graduate students working under the authors have demonstrated the advantage of utilizing fuzzy systems, neural networks, and genetic algorithms as semi-integrated processes. More complete integration of these methodologies will bring additional benefits when we learn to control the integrated fuzzy neurons and networks and the neurofuzzy systems in a straightforward manner. Indeed, the main reason for using these various methodologies in a semi-integrated (and usually sequentially) manner is to keep the processes under control.

We cannot overemphasize this last point. Neurofuzzy systems are at the state of development as neural networks before the rediscovery of backpropagation in 1968. We are proceeding on a trial-and-error basis with little guidance as to which is the best way to apply neurofuzzy concepts. The potential payoff for using neurofuzzy systems properly can be enormous. Indeed, the "fuzzy neuron" or the "neurofuzzy system" are the modern analogs of the perceptron and the adaline processing units, and they are at about the same stage of development today as the perceptron and adaline in 1960. What is now needed is the creation or discovery of an integrated training/control process.

18.4 LAST THOUGHTS

Tom Peters, coauthor of *In Search of Excellence* (Peters and Watermann, 1982), wrote the Foreword of *The Rise of the Expert Company*, an exposition on the benefits of expert systems in industry by Feigenbaum, McCorduck, and Nii (1988). The closing two paragraphs stated the following: .

> I came to this book and to the task of writing this foreword interested, even fascinated, by the topic about which I am largely naive. I leave the process of digesting the manuscript and writing the foreword mesmerized. The emerging world, brilliantly and pragmatically described in *The Rise of the Expert Company*,

> is not the world we now know. The consequences are exciting and a bit frightening—and clearly monumental.
>
> I conclude that any senior manager in any business of almost any size who isn't at least learning about AI and sticking a tentative toe or two into AI's waters is simply out of step, dangerously so.

In the eight years since that foreword was written, neural networks and fuzzy systems have achieved equally important status as expert systems in 1988. Neurofuzzy technology is the next big step because of the synergistic benefits of the merging these two important technologies. It is our hope that we have taken that first step—that is, put that first tentative toe into neurofuzzy technology's waters, lest we too get out of step, dangerously so.

REFERENCES

ACM, *Communications of the ACM*, Newstrack, Vol. 38, No. 10, pp. 11–12. 1995.

American Society for Engineering Education, *Goals of Engineering Education*, Washington, DC, 1967.

Caudill, M., *In Our Own Image: Building an Artificial Person*, Oxford Press, Cambridge, MA, 1992.

Feigenbaum, E., McCorduck, P., and Nii, H. P., *The Rise of the Expert Company*, Times Books, New York, 1988.

Peters, T. H., and Watermann, R. H., Jr., *In Search of Excellence*, Harper and Row, New York, NY, 1982.