Chapter 6

Neuro-fuzzy Systems

6.1 Introduction

This Chapter deals with neuro-fuzzy systems, i.e., those soft computing methods that combine in various ways neural networks and fuzzy concepts. Each methodology has its particular strengths and weaknesses that make it more or less suitable in a given context. For example, fuzzy systems can reason with imprecise information and have good explanatory power. On the other hand, rules for fuzzy inference have to be explicitly built into the system or communicated to it in some way; in other words the system cannot learn them automatically. Neural networks represent knowledge implicitly, are endowed with learning capabilities, and are excellent pattern recognizers. But they are also notoriously difficult to analyze: to explain how exactly they reach their conclusions is far from easy while the knowledge is explicitly represented through rules in fuzzy systems.

The complementarity between fuzzy systems and learning systems, especially ANNs, has been recognized early by researchers. Taking again a rather courageous, and utterly unrealistic biological analogy, we could say that, conceptually at least, mixed neural and fuzzy systems resemble nervous systems where neural cells are the low-level perceptive and signal integration part that make possible the higher level functions of the brain such as reasoning and linguistic abilities. In this metaphor the ANN part stands for the perceptive and signal processing biological machinery, while the fuzzy part represents the emergent "higher level" reasoning aspects. As a result, these two technologies have been integrated in various ways, giving rise to *hybrid* systems that are able to overcome many of the limitations of the individual techniques. Therefore, neuro-fuzzy systems are likely to be of wider applicability on real-life problems. The reader should be aware that the field has an enormous variety and it would be impossible to present a complete survey in a single chapter. We have thus been obliged to make a choice of topics that we believe are significant and representative of modern trends but by no means exhaustive. Two useful recent books covering in detail all the topics treated here and more are Nauck *et al.* [156] and Jang *et al.* [104].

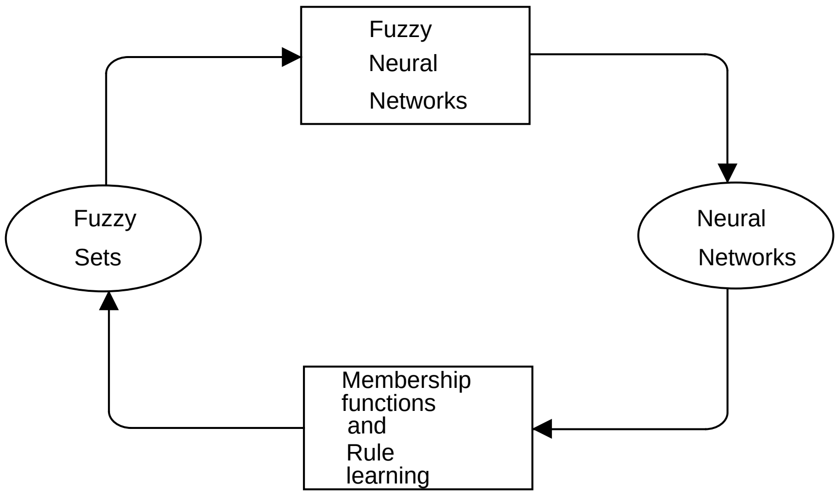

Figure 6.1 Schematic view of how artificial neural networks and fuzzy systems can interact synergetically.

There are two main ways in which ANNs and fuzzy systems can interact synergetically. One is the "fuzzification" of neural networks and the other consists in endowing fuzzy system with neural learning features. In the first case, fuzziness may be introduced at different levels in a neural net: the weight level, the transfer function level, or the learning algorithm level. In the second case, the most common arrangement is for the ANN to learn membership functions or rules for a given fuzzy system. This relationships are schematically depicted in Figure 6.1.

More precisely, according to [156], systems that combine neural and fuzzy ideas can be divided into three classes:

- *co-operative*, in the sense that neural algorithms adapt fuzzy systems, which can in turn be distinguished in
  - off-line (neural algorithms learn membership functions or rules or both, once and for all)
  - on-line (neural algorithms are used to adapt the membership functions or the rules of the fuzzy system or both, as the system operates);
- *concurrent* (but we would prefer to call them *sequential*), where the two techniques are applied after one another as pre- or post-processing.
- *hybrid* (here too, this terminology can be misleading, because, as a matter of fact, all neuro-fuzzy systems are hybrid), the fuzzy system being represented as a network structure, making it possible to take advantage of learning algorithms inherited from ANNs. From now on this combination will be called "fuzzy neural networks".

Concurrent (i.e., sequential) approaches are the weakest form of combination between neural and fuzzy techniques, and not such an interesting one for the purpose of this chapter. After all, the two techniques retain their individualities and can be understood without studying their interactions. Therefore, the first part of the chapter describes "fuzzy neural networks", that is, how single neural units and networks can be given a fuzzy flavor. The second part of the chapter deals with the other aspect of the mutual relationship that is, using ANNs to help design efficient fuzzy systems "cooperatively".

6.2 Fuzzy Neural Networks

THE PURPOSE of this section is to introduce fuzzy concepts into single artificial neurons and neural networks. Fuzzy systems and neural networks are certainly different soft computing paradigms; however, they are rather complementary if one takes into account their respective strong and weak features. Therefore, integrating them into a single new soft computing model gives hopes of exploiting their complementary nature by reinforcing the good points and by alleviating their respective shortcomings. Fuzziness can be introduced in several ways into artificial neurons. In the next section we present a way of "fuzzifying" single neurons.

6.2.1 Fuzzy Neurons

Fuzzy models of artificial neurons can be constructed by using fuzzy operations at the single neuron level. The fuzzy operations that have been used for that purpose are the union and the intersection of fuzzy sets and, more generally, *t-norms* and *t-conorms* also called *s-norms* which are extensions of the usual fuzzy operations (see Chapter 3, Section 3.2.4). A variety of fuzzy neurons can be obtained by applying fuzzy operations to connection weights, to aggregation functions or to both of them. We shall start from the structure of an artificial neuron such as it was introduced in Chapter 2 and which is reproduced here for ease of reference in Figure 6.2.

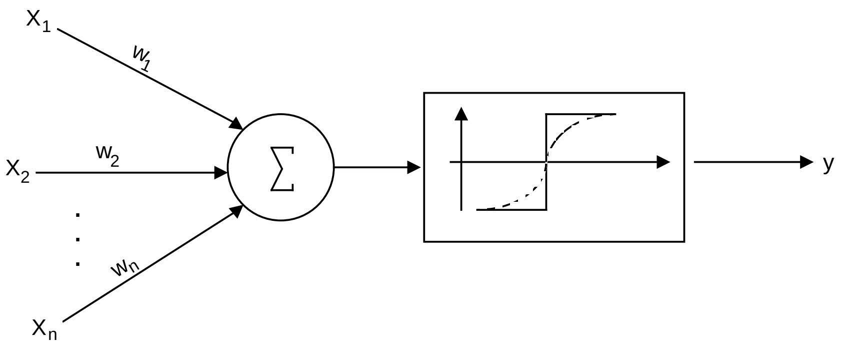

Figure 6.2 Model of a crisp artificial neuron.

We recall that a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ of input values enters a neural unit through n incoming connections after having being modified by the corresponding connection weights $\mathbf{w} = (w_1, w_2, \ldots, , w_n)$. The neuron computes the sum of the weighted inputs, which is simply the scalar product $\mathbf{w} \cdot \mathbf{x}$ and produces an output signal according to a predefined activation function g. When the function is a simple step function the neuron fires (i.e, it produces a 1 output signal) if $\mathbf{w} \cdot \mathbf{x}$ reaches a given thereshold value, otherwise it doesn't fire (the output is 0). However, for numerical reasons, it is often useful to have g as a non-linear monotonic mapping of the type:

$$g\colon [0,1] \to [0,1], \tag{6.1}$$

such as a sigmoid, hyperbolic tangent or gaussian curve. In this case the neuron emits a graded output signal between 0 and 1 (See Chapter 2, Equation 2.2 and Equation 2.3).

The above standard model of an artificial neuron derives some credibility from biological data on neural cells but it is certainly a gross oversimplification of reality. Thus, although it is a convenient choice, there is nothing special about the weighted sum of the input values as an aggregation operator. A step toward the fuzzification of an artificial neuron can be done by considering other forms A of the aggregation function according to the more general equation:

$$y = g\left(A(\mathbf{w}, \mathbf{x})\right), \tag{6.2}$$

where g is the transfer function and y is the scalar output signal of the neuron.

In fact, fuzzy union, fuzzy intersection and, more generally, *s-norms* and *t-norms* can be used as an aggregation function for the weighted inputs to an artificial neuron. Due to the fact that triangular norms form an infinite family, there exist an infinite number of possibilities for defining fuzzy neurons at the level of the aggregation function. In what follows, we present a particular class of fuzzy neurons, the OR and AND neurons.

OR Fuzzy Neuron

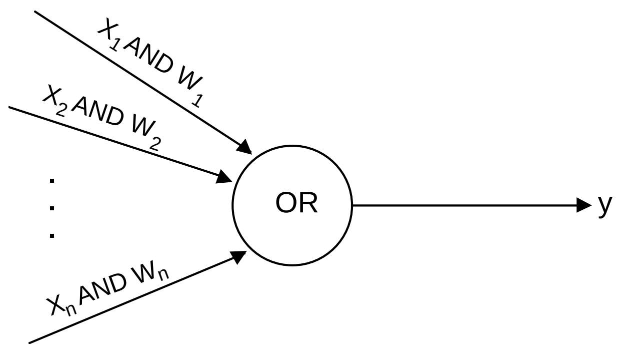

Figure 6.3 Model of an OR fuzzy neuron.

The OR fuzzy neuron realizes a mapping from the unit hypercube to fuzzy signals pertaining to the graded membership over the unit interval:

$$OR\colon [0,1] \times [0,1]^n \rightarrow [0,1]$$

the weights are also defined over the unit interval. The OR fuzzy neuron uses an aggregation function that corresponds to the maximum of the weighted inputs; that is, it selects the fuzzy disjunction of the weighted inputs as follows:

$$y = OR(x_1\ AND\ w_1, x_2\ AND\ w_2, \ldots, x_n\ AND\ w_n). \qquad (6.3)$$

This setting is depicted in Figure 6.3. As we saw in Chapter 3, Section 3.2.4, fuzzy set logical connectives are usually defined in terms of triangular norms $\triangle$ and t-conorms $\bigtriangledown$. Thus, the preceding expression for the neuron output becomes:

$$y = \bigtriangledown_{j=1}^{n}(x_j\ \triangle\ w_j). \qquad (6.4)$$

The transfer function g is linear. In a manner analogous to standard ANNs a bias term can be added representing a constant $x_0 = 0$ value input signal with a weight w_0. Taking the bias into account the preceding equation reads:

$$y = \bigtriangledown_{j=0}^{n}(x_j\ \triangle\ w_j). \qquad (6.5)$$

We observe that, for any connection k, if $w_k = 0$ then $w_k\,AND\,x_k = 0$ while if $w_k = 1$ then $w_k\,AND\,x_k = x_k$ independent of x_k.

AND Fuzzy Neuron

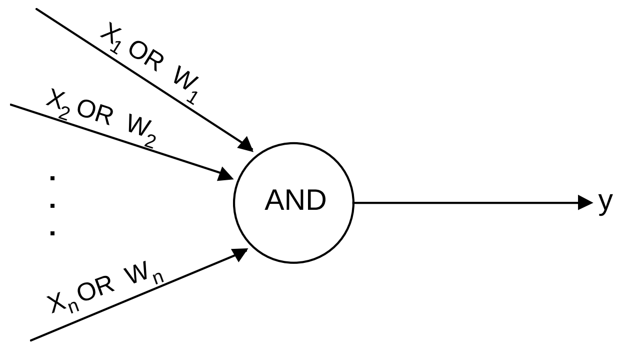

Figure 6.4 Model of an AND fuzzy neuron.

The AND fuzzy neuron (see Figure 6.4) is similar to the OR case except that it takes the fuzzy conjunction of the weighted inputs, which is the same as the minimum. First the inputs are "ored" with the corresponding connection weights and then the results are aggregated according to the AND operation. The transfer function g is again linear:

$$y = AND(x_1\,OR\,w_1, x_2\,OR\,w_2, \ldots, x_n\,OR\,w_n). \qquad (6.6)$$

Analogous to the OR case, when expressed in terms of triangular norms the output fuzzy value is given by:

$$y = \triangle_{j=0}^{n}(x_j \ \triangledown \ w_j), \tag{6.7}$$

where the bias term x_0 is now equal to 1, giving $w_0 \ AND \ x_0 = w_0$.

Of course, in the generalized forms based on t-norms, operators other than *min* and *max* can be used such as algebraic and bounded products and sums. As they stand, both the OR and the AND logic neurons are *excitatory* in character, that is higher values of x_k produce higher values of the output signal y. The issue of *inhibitory* (negative) weights deserves a short digression since their introduction is not as straightforward as it is in the standard neural networks. In fact, we are here in the realm of fuzzy sets and we would obviously like to maintain the full generality of the operations defined in $[0,1]$. If the interval is extended to $[-1,1]$ as it is customary in ANNs, logical problems arise from the fuzzy set-theoretical point of view. The proper solution to make a weighted input inhibitory is to take the fuzzy complement of the excitatory membership value $\bar{x} = 1 - x$. In this way, the generic input vector $\mathbf{x} = (x_1, x_2 \ldots x_n)$ now includes the complemented values as well: $\mathbf{x} = (x_1, x_2 \ldots x_n, \bar{x}_1, \bar{x}_2 \ldots \bar{x}_n)$. The weighted inputs $x_i \ \circ \ w_i$, where $\circ$ is a t-norm or t-conorm, can be general fuzzy relations too, not just simple products as in standard neurons. As well, the transfer function g, which has been supposed linear, can be a non-linear one such as a sigmoid. These kinds of fuzzy neurons have been introduced by Pedrycz and coworkers [173, 195].

OR/AND Fuzzy Neuron

A generalization of the above simple fuzzy neurons is the OR/AND neuron [95]. The OR/AND neuron is a combination of the AND and OR neurons into a two-layer structure as depicted in Figure 6.5. Taken as a whole, this structure can produce a spectrum of intermediate behaviors that can be modified in order to suit a given problem. Looking at the figure, it is apparent that the behavior of the net can be modulated by suitably weighting the output signals from the OR or the AND parts through setting or learning the connection weights c_1 and c_2. The limiting cases are $c_1 = 0$ and $c_2 = 1$ where the system reduces itself to a pure AND neuron and the converse $c_1 = 1$, $c_2 = 0$, in which case the behavior corresponds to that of a pure OR neuron.

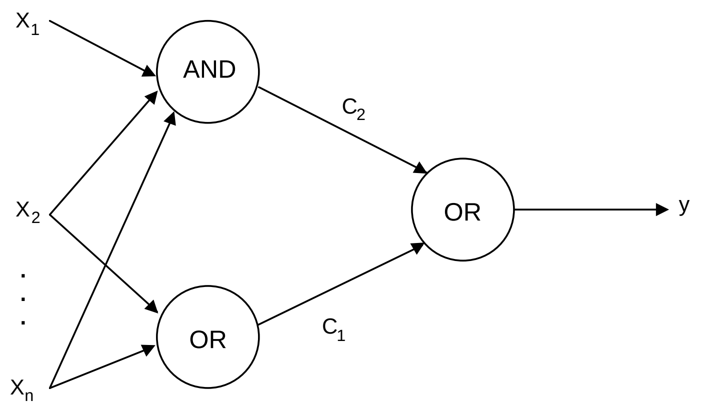

Figure 6.5 The computational structure of an AND/OR fuzzy neuron.

6.2.2 Multilayered Fuzzy Neural Networks

Fuzzy neurons such as those described in the previous section can be assembled together into multilayered networks [170]. Since the neurons used to build the nets are in general different, the construction gives rise to non-homogeneous neural networks, in contrast with the usually homogeneous networks that are used in the crisp ANN domain. For example, Figure 6.6 depicts a two-layer network (not counting the input layer) composed of a first layer with p neurons of the same AND type and a second output layer wich aggregates all the preceding signals with a single OR neuron. The input is constituted by $2n$ values including both the direct and the complemented ones. A second possibility is to have OR neurons in the hidden layer and a single AND neuron in the output layer (Figure 6.7). These two types of networks have been called *logic processors* by Pedrycz.

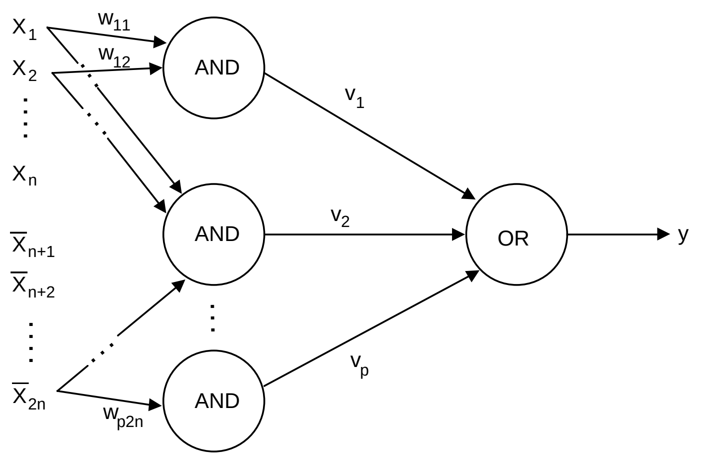

Figure 6.6 A three-layer artificial neural network with all AND fuzzy units in the hidden layer.

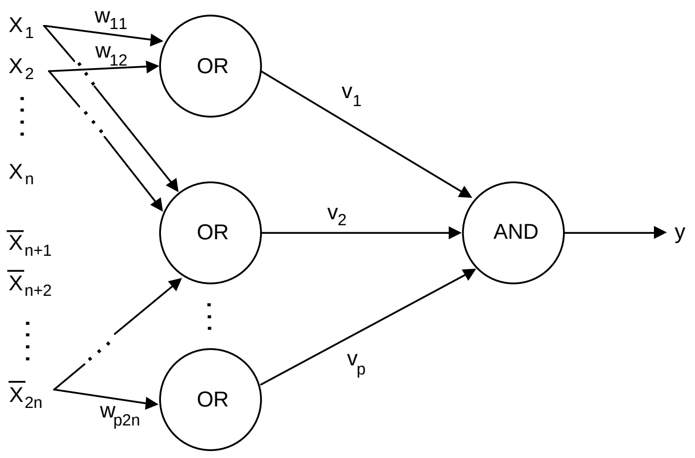

Figure 6.7 A three-layer artificial neural network with all OR fuzzy units in the hidden layer.

Following Pedrycz [170], for the first layered network type we have that the signals z_k produced at the hidden layer are given by:

$$z_k = [\triangle_{j=1}^{n}(x_j \;\triangledown\; w_{k,j})] \;\triangle\; [\triangle_{j=1}^{n}(\bar{x}_j \;\triangledown\; w_{k,(n+j)})], \quad k = 1, 2, \ldots, p; \tag{6.8}$$

where $\mathbf{w}_k$ is the vector of all connection weights from the inputs to the kth node of the hidden layer. The output value y is the single OR-aggregation of the previous signals from the hidden layer:

$$y = \triangledown_{j=1}^{p}(x_j \;\triangle\; v_j). \tag{6.9}$$

In the last expression the vector $\mathbf{v}$ is the weight vector of all the connections from the hidden layer nodes to the single output node. The network in which all the hidden nodes are of the OR type and the output node is an AND fuzzy neuron (Figure 6.7) gives analogous expressions with the $\triangle$ and $\triangledown$ symbols exchanged. If we restrict ourselves to the pure two-valued Boolean case then the network with the hidden OR layer represents an arbitrary Boolean function as a sum of minterms, while the second network is its dual in the sense that it represents any Boolean function as a product of maxterms. More generally, if the values are continuous members of a fuzzy set then these networks approximate a certain unknown fuzzy function.

6.2.3 Learning in Fuzzy Neural Networks

The interest of having fuzzy connectives organized in network form is that there are thus several ways in which ANN supervised learning methods can be applied to the fuzzy structure. This is a definite plus in many situations since learning capabilities are not typical of fuzzy systems. Of course, there exist other ways in which fuzzy systems can learn but this particular neuro-fuzzy hybrid is useful in light of the large amount of knowledge that has been accumulated on the crisp ANNs versions. Supervised learning in fuzzy neural networks consists in modifying their connection weights in a such a manner that an error measure is progressively reduced by using sets of known input/output data pairs. Another important requirement is that the network thus obtained be capable of generalization; that is, its performance should remain acceptable when it is presented with new data (see the discussion on ANN supervised learning in Chapter 2, Section 2.5).

Let us call the set of n training data pairs $(\mathbf{x}_k, d_k)$ for $k = 1, 2 \ldots n$, where $\mathbf{x}_k$ is a vector of input data and d_k is the corresponding observed scalar output. A single fuzzy neuron adapts its connection weights in order to reduce a measure of error averaged over the training set:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \Delta\mathbf{w}^t, \tag{6.10}$$

where the weight change is a given function F of the difference between the target response d and the calculated node output y:

$$\Delta\mathbf{w}^t = F(|d^t - y^t|), \tag{6.11}$$

For instance, in standard ANNs a common learning rule is the *delta* rule, which uses an estimate of the gradient of the continuous neuron activation to reduce the mean square error (Chapter 2, Section 2.5). For fuzzy neurons, one should take into account the fact that the weighted inputs are not simple scalar products; rather, they are more complex relationships between fuzzy sets.

For the whole network supervised learning proceeds as follows. A criterion function E is defined such that it gives a mesure of how well the fuzzy network maps input data into the corresponding output. A common form for E is the sum of the squared errors:

$$E(\mathbf{w}) = 1/2 \sum_{k=1}^{n} (d_k - y_k)^2 \tag{6.12}$$

The goal of the learning algorithms is to systematically vary the connection weights in such a way that E is minimized on the training data set. This can be achieved by taking the gradient descent of E with respect to the weights. The update step in the output weights $w_{i,j}$ of the connections between unit j in the hidden layer and unit j in the output layer can thus be found by differentiating:

$$\Delta w_{i,j} = -\eta \, \frac{\partial E}{\partial w_{i,j}}, \tag{6.13}$$

where η is a scale factor that controls the magnitude of the change. The update in the input weights can be found by using the chain rule for partial derivatives in the backpropagation style (see also Chapter 2, Section 2.5). The derivation of the weight changes layer by layer back to the input layer is straightforward but somewhat tricky. The interested reader can find an example completely worked out in Pedrycz's book [170]. The network chosen for the example corresponds to the three-layer system of Equation 6.8 and Equation 6.9 where the algebraic sum is used for the s-norm and the product for the t-norm.

6.2.4 An Example: NEFPROX

Approximating a continuous unknown function specified by sample input/output data pairs is a widespread problem. We already saw in Chapter 2 how multilayer neural networks can implicitly approximate such a mapping. Here we present another approach to this problem by using a neuro-fuzzy system. The discussion that follows is based on the work of D. Nauck [155, 157].

In this approach, called NEFPROX for NEuro Fuzzy function apPROXimator, a neuro-fuzzy systems is seen as a three-layer feedforward network similar to the type described in the preceding section. There are no cycles in the network and no connections exist between layer n and layer $n + j$, with $j > 1$. The first layer represents input variables, the hidden layer represents fuzzy rules, and the third layer represents output variables. The hidden and output units in this network use t-norms and t-conorms as aggregation functions, in a manner similar to what we have seen in the previous sections. Fuzzy sets are encoded as fuzzy connection weights and fuzzy inputs. The whole network is capable of learning and provides a fuzzy inference path. The end result should be interpretable as a system of linguistic rules.

The problem to be solved is that of approximating an unknown continuous function using a fuzzy system given a set of data samples. There is an existence proof that fuzzy systems are capable of universal function approximation [116]. However, actually building such an approximation for a given problem requires the specification of parameters under the form of membership functions and of a rule base. This identification can be done by previous knowledge, trial and error, or by some automatic learning methodology. NEFPROX encodes the problem parameters in the network and uses a supervised learning algorithm derived from neural network theory in order to drive the mapping towards satisfactory solutions. The advantage of the fuzzy approach over a standard neural network is that, while the latter is a black box, the fuzzy system can be interpreted in terms of rules and thus has more descriptive power.

The NEFPROX system is a three-layer network with the following features:

- The input units are labeled $x_1, x_2, \ldots, x_n$. The hidden rule units are called $R_1, R_2, \ldots, R_k$ and the output units are denoted as $y_1, y_2, \ldots, y_m$.
- Each connection is weighted with a fuzzy set and is labeled with a linguistic term.
- All connections coming from the same input unit and having the same label are weighted by the same common weight, which is called a *shared* weight. The same holds for the connections that lead to the same output unit.
- There is no pair of rules with identical antecedents.

According to these definitions, it is possible to interpret a NEFPROX system as a fuzzy system in which each hidden unit stands for a fuzzy if-then rule. Shared weights are needed in order for each linguistic value to have a unique interpretation. If this were not the case, it would be possible for fuzzy weights representing identical linguistic terms to evolve differently during learning, leading to different individual membership functions for its antecedents and conclusions variables, which would in turn prevent proper interpretation of the fuzzy rule base. Figure 6.8 graphically depicts the structure of a NEFPROX system.

Learning in NEFPROX is based on supervised training and employs a conceptual variation of standard backpropagation in ANNs

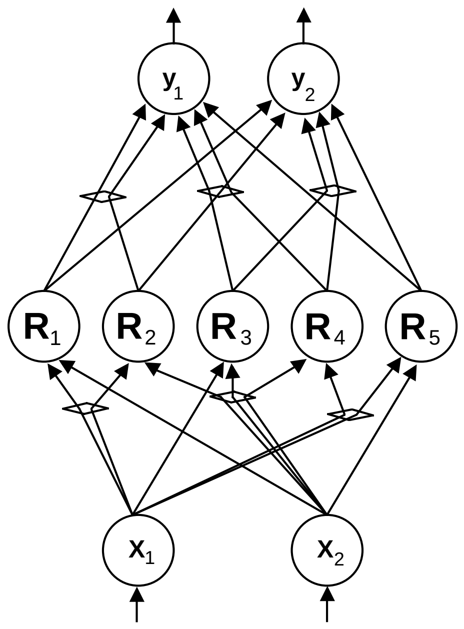

Figure 6.8 Schematic architecture of a NEFPROX system. The connections going through the small rhombuses are linked i.e., they share the same weight (see text). The figure is adapted from Nauck [155].

since we are in a framework where known input/output sets do usually exist. The difference with the standard algorithm is that the method for determining the errors and propagating them backwards to effect local weight modifications is not based on gradient descent. This is due to the fact that the functions involved in the system are not always differentiable, as is the case for some types of triangular norms such as minimum and maximum used here. Central to the NEFPROX approach to learning is simplicity, speed, and interpretability of the results. The system is more suitable for Mamdani-type fuzzy systems with a small number of rules and a small number of meaningful membership functions. Indeed, according to [157], if very precise function approximation is called for, then a neuro-fuzzy approach, which should be characterized by tolerance for imprecision, is probably not well suited anyway and other methodologies should be preferred.

Since fuzzy rules are used in NEFPROX to approximate the unknown function, pre-existing knowledge can be used at the outset by initializing the system with the already known rules, if any. The remaining rules have to be found in the learning process. If nothing is known about the problem, the system starts out without hidden units, which represent rules, and incrementally learns them. This constructive aspect of the algorithm constitutes another difference with respect to the usual backpropagation learning algorithm in a fixed network architecture. Simple triangular membership functions are used for fuzzy sets although other forms would also be permissible. At the beginning of the learning process fuzzy partitions for each input variable are specified. Fuzzy sets for output variables are created during learning and a defuzzyfication procedure is used at the output nodes to compare calculated and observed values. The network structure is as described above (see also Figure 6.8). Given a training set of patterns $\{\mathbf{s}_1, \mathbf{t}_1, \ldots, \mathbf{s}_r, \mathbf{t}_r\}$ where $\mathbf{s} \in \mathbb{R}^n$ is an input pattern and $\mathbf{t} \in \mathbb{R}^m$ the desired output, the learning algorithm has two parts: a structure-learning part and a parameter-learning part. The following is a slightly simplified description of the algorithm, more details can be found in the original work [157].

Structure Learning Algorithm

1. Select the next training pattern $(\mathbf{s}, \mathbf{t})$ from the training set.

2. For each input unit x_i find the membership function $\mu_{ji}^{(i)}$ such that
$$\mu_{ji}^{(i)}(s_i) = \max_{j \in \{1,\ldots,p_i\}} \{\mu_j^{(i)}(s_i)\}.$$

3. If there is no rule R with weights $W(x_1, R) = \mu_{ji}^{(1)}, \ldots, W(x_n, R) = \mu_{jn}^{(1)}$ then create the node and connect it to all the output nodes.

4. For each connection from the new rule node to the output nodes find a suitable fuzzy weight $\nu_{ji}^{(i)}$ using the membership functions assigned to the output units y_i such that $\nu_{ji}^{(i)}(t_i) = \max_{j \in \{1,\ldots,q_i\}} \{\nu_j^{(i)}(t_i)\}$ and $\nu_j^{(i)}(t_y) \geq 0.5$. If the fuzzy set is not defined then create a new one $\nu_{\text{new}}^{(i)}(t_i)$ for the output variable y_i and set $W(R, y_i) = \nu_{\text{new}}^{(i)}$.

5. If there are no more training patterns then stop rule creation; otherwise go to 1.

6. Evaluate the rule base and change the rule conclusions if appropriate.

The supervised learning part that adapts the fuzzy sets associated to the connection weights works according to the following schema:

Parameter Learning Algorithm

1. Select the next training pattern $(\mathbf{s}, \mathbf{t})$ from the training set and present it at the input layer.

2. Propagate the pattern forward through the hidden layer and let the output units determine the output vector $\mathbf{o}$.

3. For each output unit y_i determine the error $\delta_{yi} = t_i - o_{yi}$.

4. For each rule unit R with output $o_R > 0$ do:

   - Update the parameters of the fuzzy sets $W(R, y_i)$ using a learning rate parameter $\sigma > 0$.
   - Determine the change $\delta_R = o_R(1 - o_R) \cdot \sum_{y \in \text{output layer}} (2W(R,y)(t_i) - 1) \cdot |\delta_y|$.
   - Update the parameters of the fuzzy sets $W(x, R)$ using δ_R and σ to calculate the variations.

5. If a pass through the training set has been completed and the convergence criterion is met then stop; otherwise go to step 1.

The learning procedure for the fuzzy sets is based on simple heuristics that result in shifting the membership functions and in making their support larger or smaller. It is possible and easy to impose constraints on the learning procedures such as that fuzzy sets must not pass each other or that they must intersect at some point and so on. As usual in supervised learning algorithms, one or more validation sets of data are used and training goes on until the error on the validation set starts to increase in order to avoid overfitting and to promote generalization (see also Chapter 2, Section 2.5).

NEFPROX has been tested on a well-know difficult benchmark problem: the Mackey-Glass system. The Mackey-Glass delay-differential equation was originally proposed as a model of white blood cell production:

$$\frac{dx}{dt} = \frac{0.2x(t-\tau)}{1 + x^{10}(t-\tau)} - 0.1x(t), \tag{6.14}$$

where τ is a parameter. Using a value of $\tau = 17$ the resulting series is chaotic. Training data can be obtained by numerical integration of the equation. A thousand values were calculated of which the first half were used for training and the rest for validation. The NEFPROX system used to approximate the time series has four input and one output variable and each variable was initially partitioned by seven equally distributed fuzzy sets with neighboring membership functions intersecting at degree 0.5. After learning, 129 fuzzy rules were created. The resulting system approximates the function quite well in the given range. The results are only slightly worse than those that have been obtained on the same problem with another neuro-fuzzy system called ANFIS [103] (see next section) but the learning time is much shorter.

Two related neuro-fuzzy approaches are NEFCON and NEFCLASS which are used, respectively, for control applications and for classification problems [155]. NEFCON is similar to NEFPROX but has only one output variable and the network is trained by reinforcement learning using a rule-based fuzzy error measure as a reinforcement signal. NEFCLASS sees pattern classification as a special case of function approximation and uses supervised learning in a manner similar to NEFPROX to learn classification rules.

6.2.5 A Second Example: The ANFIS System

ANFIS stands for Adaptive Network-based Fuzzy Inference System and is a neuro-fuzzy system that can identify parameters by using supervised learning methods [103]. ANFIS can be thought of as a network representation of Sugeno-type fuzzy systems with learning capabilities. ANFIS is similar in spirit to NEFPROX but, with respect to the latter, learning takes place in a fixed structure network and it requires differentiable functions. The ANFIS heterogeneous network architecture is constituted by a number of layers of nodes which have the same function for a given layer but are different from one layer to the next. For example, consider the fuzzy inference system with two inputs x and y and a single output z [103]. For a first-order Sugeno model, a rule set using a linear combination of the inputs can be expressed as:

$$\begin{array}{l} \text{IF } x \text{ is } A_1 \text{ AND } y \text{ is } B_2 \text{ THEN } f_1 = p_1 x + q_1 y + r_1 \\ \text{IF } x \text{ is } A_2 \text{ AND } y \text{ is } B_2 \text{ THEN } f_2 = p_2 x + q_2 y + r_2 \end{array} \tag{6.15}$$

The reasoning mechanism for this model is:

$$f = \frac{w_1 f_1 + w_2 f_2}{w_1 + w_2} = \bar{w}_1 + \bar{w}_2. \tag{6.16}$$

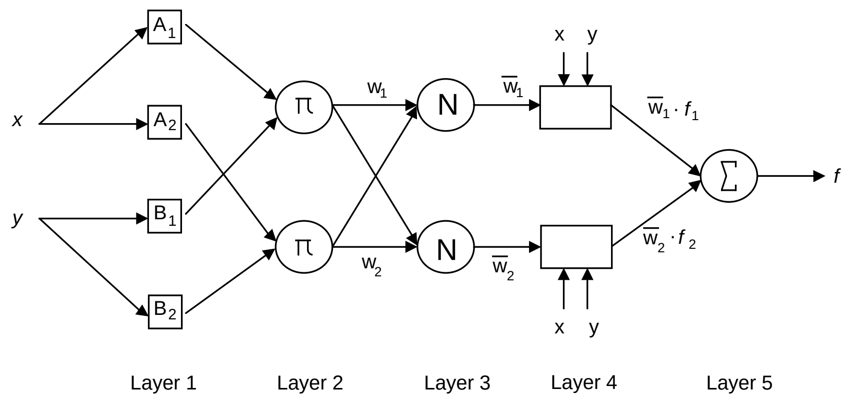

Figure 6.9 ANFIS architecture corresponding to a two-input first-order Sugeno fuzzy model with two rules (see text). The figure is adapted from the work of Jang and Sun [103].

The ANFIS network architecture corresponding to this Sugeno model is shown in Figure 6.9. The layers in the net are constituted by nodes having the same function for a given layer. The functionalities of the layers are as follows:

Layer 1: Denoting by $O_{l,i}$ the output of node i in layer l, each node in layer 1 is an adaptive unit with output given by:

$$\begin{aligned} O_{1,i} &= \mu_{A_i}(x), \quad i = 1, 2 \\ O_{1,i} &= \mu_{B_{i-2}}(x), \quad i = 3, 4 \end{aligned} \tag{6.17}$$

where x and y are input values to the node and A_i or B_{i-2} are fuzzy sets associated with the node. In other words, each node in this layer generates the membership grades of the premise part. The membership functions for A_i and B_i can be any appropriate parameterized membership function such as triangular, trapezoidal, Gaussian or bell-shaped.

Layer 2: Each node in this layer is labeled Π and computes the firing strength of each rule as the product of the incoming inputs or any other t-norm operator:

$$O_{2,i} = w_i = \mu_{A_i}(x) \bigtriangleup \mu_{B_i}(y), \quad i = 1, 2. \tag{6.18}$$

Layer 3: Each node in this layer is labeled N and it calculates the ratio of the i-th rule's firing strength to the sum of all rules' firing strengths:

$$O_{3,i} = \bar{w}_i = \frac{w_i}{w_1 + w_2}, \quad i = 1, 2. \tag{6.19}$$

Layer 4: Each node in this layer has the following function:

$$O_{4,i} = \bar{w}_i f_i = \bar{w}_i (p_i x + q_i y + r_i), \tag{6.20}$$

where $\bar{w}_i$ is the output of layer 3 and $\{p_i, q_i, r_i\}$ is the parameter set (see Equation 6.15).

Layer 5: There is a single node Σ in this layer. It aggregates the overall output as the summation of all the incoming signals:

$$O_{5,1} = \sum_i \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{6.21}$$

This completes the construction of the network which is seen to have the same functionality as the equivalent Sugeno model.

Learning in ANFIS

The ANFIS learning algorithm is a hybrid supervised method based on gradient descent and least-squares methods. In the forward phase, signals travel forward up to layer 4 and the relevant parameters are fitted by least-squares. In the backward phase the error signals travel backward and the premise parameters are updated as in backpropagation. More details of the algorithm can be found in [103]. It is worth noting that the ANFIS network with its learning capabilities can be built by using the fuzzy toolbox available in the MATLAB package.

Function Modeling and Time Series Prediction

ANFIS can be applied to non-linear function modeling and time series prediction. ANFIS gives excellent results on the prediction of the time series generated by the numerical integration of the Mackey-Glass delay-differential equation prediction of the time series generated by the numerical integration of this equation, better than most other approaches for function approximation such as those based on neural networks of various types [103] and on standard function fitting methods. The ANFIS system shows excellent non-linear fitting and generalization capabilities on this example. As well, the number of parameters and the training time is comparable or less than what is required by ANN methods, with the exception of the neuro-fuzzy system NEFPROX, which learns faster and has slightly fewer parameters, as we saw above.

ANFIS for Neuro-Fuzzy Control

Fuzzy control has been introduced in Chapter 3, Section 5.2.4. The time evolution of a dynamical system can be described by the following differential equation:

$$\frac{d\mathbf{x}}{dt} = F(\mathbf{x}, \mathbf{u}),$$

where $\mathbf{x}$ represents the state of the system and $\mathbf{u}$ is a vector of controllable parameters. The control action is formally given by a function g that maps the system state into appropriate parameters for a given control problem:

$$\mathbf{u}(t) = g(\mathbf{x}(t)).$$

We have seen in Chapter 3, Section 5.2.4 that the problem of finding optimal control policies for non-linear systems is mathematically very difficult, while fuzzy approaches have proved effective in many cases. Since a wide class of fuzzy controllers can be transformed into equivalent adaptive networks, ANFIS can be used for building intelligent controllers that is, controllers that can reason with simple fuzzy inference and that are able to learn from experience in the ANN style.

6.3 "Co-operative" Neuro-fuzzy Systems

Another level of integration between artificial neural networks and fuzzy systems tries to take advantage of the array of adaptation and learning algorithms devised for the former to tune or create all or some aspects of the latter, and *vice versa*.

One important thing to note is that such approaches refrain from casting the fuzzy system into a network structure, or fuzzifying the elements of the neural network, unlike other approaches discussed in Section 6.2.

One could also note that, under this perspective, radial-basis function networks, a type of neural network with bell-shaped activation functions instead of sigmoid, might be interpreted as neuro-fuzzy networks in their own way, simply by considering their activation functions as membership functions.

6.3.1 Adaptive Fuzzy Associative Memories

A possible interpretation of a fuzzy rule, proposed by Kosko [115], views it as an association between antecedent and consequent variables. Kosko calls a fuzzy rule base complying with that semantic interpretation a *fuzzy associative memory* (FAM).

Associative Memories

An associative memory consists of memory components of the form (k, i), where k is a key and i the information associated with it. Retrieval of a memory component depends only on its key and not on its place in the memory. Recall is done by presenting a key k^*, which is simultaneously compared to the keys of all memory components. The information part i^* is found (or reported missing) within one memory cycle.

Associative memories can be implemented as neural networks, and in that case one speaks of neural associative memories: if a key pattern is presented to a neural associative memory, the activations of the output units represent the corresponding information pattern.

Fuzzy Associative Memories

When a variable x takes up values in a finite discrete domain $X = \{x_1, \ldots, x_m\}$, a fuzzy set A with membership function $\mu_A\colon X \to [0,1]$ can be viewed as a point $\mathbf{v}_A$ in the m-dimensional hypercube, identified by the co-ordinates

$$\mathbf{v}_A = (\mu_A(x_1), \ldots, \mu_A(x_m)).$$

Accordingly, a fuzzy rule R of the form

$$\text{IF } x \text{ is } A \text{ THEN } y \text{ is } B$$

can be viewed as a function mapping $\mathbf{v}_A$ (a point in $[0,1]^m$) to $\mathbf{v}_B$ (a point in the hypercube, say $[0,1]^s$ defined by the domain Y of y).

A fuzzy associative memory is a two-layer network, with one input unit for each discrete value x_i in every domain X of input variables and one output unit for each discrete value y_j in the output variable domain Y. Activation for all units can range in $[0,1]$ and is to be interpreted as the degree of membership of the relevant discrete value in the relevant linguistic value. The weights between input unit-output unit pairs can range in $[0,1]$ and the activation function for output unit u_j is

$$u_j = \mathop{\triangledown}_{i=1,\ldots,m} v_i \bigtriangleup w_{ij}. \tag{6.22}$$

A FAM is determined by its connection weight matrix $\mathbf{W} = (w_i j)$, with $i = 1, \ldots, m$ and $j = 1, \ldots, s$. Such a FAM stores just one rule. Matrix $\mathbf{W}$ is called *fuzzy Hebb matrix*.

Given an input fuzzy set A in the form of a vector $\mathbf{v}_A$ and the corresponding output fuzzy set B in the form of a vector $\mathbf{u}_B$, the fuzzy Hebb matrix storing their association is given by the *correlation minimum encoding* [115]

$$\mathbf{W} = \mathbf{v} \circ \mathbf{u}, \quad w_{ij} = v_i \bigtriangleup u_j. \tag{6.23}$$

The associative recall is given by

$$\mathbf{u} = \mathbf{v} \circ \mathbf{W}, \quad u_j = \mathop{\triangledown}_{i=1,\ldots,m} v_i \bigtriangleup w_i j. \tag{6.24}$$

The recall is always correct if $h(\mu_A) \geq h(\mu_B)$, where $h(\cdot)$ is the height of a membership function, i.e., the maximum degree of membership. If we restrict our attention to normal fuzzy sets, then the recall will always be correct.

Summarizing, the concept of a FAM should be nothing really new to the reader. Once the notational details are clear, one can recognize that a FAM is simply a matrix-vector representation of a fuzzy relation or a fuzzy rule, and one which resembles very closely two-layer neural networks.

FAM Systems

Because combination of multiple fuzzy Hebb matrices into a single matrix is not recommended lest a severe loss of information is incurred, each rule of a fuzzy rule base should be represented by a distinct FAM. The overall output of the system is then given by the component-wise maximum of all FAM outputs. Such a fuzzy system, shown in Figure 6.10, is called a *FAM system*.

The FAM system is completed by a fuzzification and a defuzzification component, and by weights associated with each FAM.

One strength of a FAM's striking resemblance with a two-layer artificial neural network is that we can borrow some learning techniques and make FAMs adaptive.

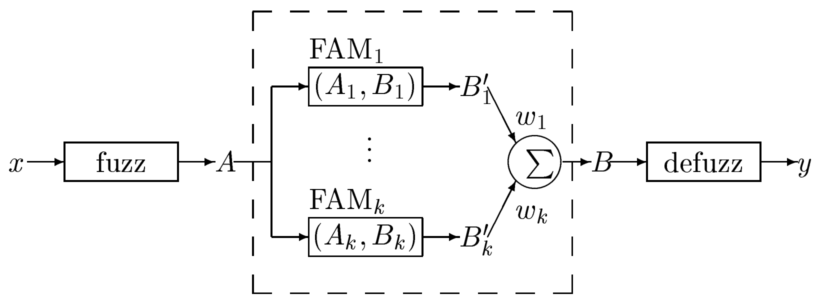

Figure 6.10 General scheme of a FAM system.

Learning in Adaptive FAMs

Kosko suggests two approaches to learning for adaptive FAMs. The first possibility is to learn the weights associated with FAMs' outputs. The second and more interesting is to create FAMs completely by learning.

The learning procedure proposed by Kosko is *differential competitive learning*, a form of adaptive vector quantization (see Chapter 2, Section 2.6.2).

Given a data set of examples of inputs and correct output values, with n input variables and one output variable, the idea is to build a two-layer network with as many input units as variables (i.e., $n + 1$) and as many output units as the number of possible rules one could build from the given variable and a predefined partition of their domains (that is, the user must determine the linguistic values in advance). To begin with, all the input units are connected to all the output units and the output units are completely connected by inhibitory links. The examples in the data set form clusters in the product space of all variables; the learning process is supposed to develop prototypes of these clusters; each cluster is interpreted as an instance of a fuzzy rule and their best matching prototypes are selected by the learning process. Therefore, the learning procedure selects the FAMs to be included in the system and assigns them a weight; if further training data are collected during the use of the FAM system, the learning process can resume and continue concurrently with the operation of the system, by updating the rule weights or by deleting or adding rules.

6.3.2 Self-Organizing Feature Maps

An approach similar to Kosko's differential competitive learning is proposed by Pedrycz and Card [171]. They use a self-organizing feature map (cf. Chapter 2, Section 2.6.3) with a planar competition layer to cluster training data, and they provide means to interpret the result of learning as linguistic rules.

The self-organizing map has an input layer with n units, where n is the number of variables in a dataset record. The output layer of the map is a $n_1 \times n_2$ lattice of units. Inputs and connection weights are in $[0,1]$. It is convenient to specify the connection weights between input and output units as a three-dimensional matrix $\mathbf{W} = (w_{i_1,i_2,i})$, where $i_1 = 1,\ldots,n_1$, $i_2 = 1,\ldots,n_2$, and $i = 1,\ldots,n$.

The result of learning a set of sample records (or vectors) $\mathbf{x}_k = (x_{k1},\ldots x_{kn})$, $k = 1,\ldots,m$, shows whether two input records are similar, i.e., belong to the same class. However, if n is sufficiently large, the structure of the problem is not usually detected in the two-dimensional map. Rather, Pedrycz and Card provide a procedure for interpreting the result using linguistic variables.

After learning, each variable x_i can be described by a matrix $\mathbf{W}_i$, which contains the weights of the connections between the relevant input unit u_i and all the output units. This constitutes the map for a single variable, or feature. The procedure consists in specifying a number of fuzzy sets $A^{(i)}_{j_i}$ for each variable x_i, with membership function $\mu_{A^{(i)}_{j_i}}$. These membership functions are applied to matrix $\mathbf{W}_i$ to obtain an equal number of transformed matrices $\mu_{A^{(i)}_{j_i}}(\mathbf{W}_i)$. The transformed matrices have higher values in those areas of the map that are compatible with the linguistic concept represented by $A^{(i)}_{j_i}$.

Each combination of linguistic terms is a possible linguistic description of a cluster of records from the data set. To check a linguistic description for validity, the transformed matrices are intersected, yielding a matrix $\mathbf{D} = (d_{i_1,i_2})$, which can be interpreted as a fuzzy relation among the variables:

$$\mathbf{D} = \bigwedge_{i=1}^{n} \mu_{A^{(i)}_{j_i}}(\mathbf{W}_i), \quad d_{i_1,i_2} = \min_{i=1,\ldots,n} \{\mu_{A^{(i)}_{j_i}}(w_{i_1,i_2,i})\}. \tag{6.25}$$

Each linguistic description is a valid description of a cluster if the relevant fuzzy relation $\mathbf{D}$ has a non-empty α-cut $\mathbf{D}_\alpha$. If the variables are separated into input and output variables, according to the particular problem at hand, then each valid linguistic description readily translates into an IF-THEN rule.

Compared to Kosko's FAMs, Pedrycz and Card's approach is more computationally expensive, because all combinations of linguistic terms must be examined in order to produce the desired fuzzy rule base. Furthermore, the determination of a sufficiently high threshold α for assessing description validity and of the right number of neurons in the output layer is a problem that has to be individually solved for every learning problem. However, the advantages are that the rules are not weighted and that the user-defined fuzzy sets have a guiding influence on the learning process, leading to more interpretable results.

6.3.3 Learning Fuzzy Sets for Sugeno-Type Fuzzy Systems

A method for learning the fuzzy sets in a Sugeno-type fuzzy rule-based system using supervised learning is the one proposed by Nomura and colleagues [169].

First of all, the assumption is made that linguistic values referred to by rule antecedents are defined by parameterized triangular fuzzy numbers, that is, membership functions of the form

$$\mu(x) = \begin{cases} 1 - 2\frac{|x-C|}{b} & C - \frac{b}{2} \leq x \leq C + \frac{b}{2} \\ 0 & \text{otherwise,} \end{cases} \tag{6.26}$$

where C is the center and b is the base, or width, of the triangle. The consequent of a rule just consists of a crisp value w_0 (this is a degenerate case of the Sugeno model). The product is used as the t-norm.

The learning algorithm is based on gradient descent using the half squared error as the error measure:

$$\text{hse} = \frac{1}{2}\sum_{k=1}^{m}(y_k - y_k^*)^2, \tag{6.27}$$

where y_k is the value computed by the fuzzy system and y_k^* is the actual value in the training data set. Since the type of Sugeno model adopted applies only differentiable operations, both to determine the degree of truth of the antecedents (product) and to aggregate the outputs of all the rules (weighted average), the calculation of the changes for the parameters C and b of each membership function and w_0 of each rule is equivalent to the generalized delta rule (cf. Chapter 2, Section 2.5) for multilayer neural networks.

The only caution one must have is that the triangular membership functions are not differentiable in three points. However, it is not too difficult to devise satisfactory heuristics that overcome this potential problem.

One disadvantage of this approach is that the semantics of the linguistic values depend on the rules they appear in. Whereas in the initial (hand-crafted) rule base identical linguistic terms are described by distinct yet identical membership functions, the learning procedure changes this state of affairs, by modifying the membership functions of each term independently of the others. Such effect is undesirable, for it obfuscates the interpretation of the resulting rule base. A way to overcome this difficulty, proposed by Bersini, Nordvik, and Bonarini [27] would be to make identical linguistic terms share the same membership function.

6.3.4 Fuzzy ART and Fuzzy ARTMAP

A Fuzzy ART (for "Adaptive Resonance Theory") neural network [41] is a self-organizing neural network capable of clustering collections of arbitrarily complex analog input patterns via unsupervised learning.

Fuzzy ART Neural Networks

The Fuzzy ART neural network architecture, illustrated in Figure 6.11, consists of two subsystems, the *attentional* subsystem and the *orienting* subsystem. The attentional subsystem consists of two layers L_1 and L_2. L_1 is called the *input* layer because input patterns are applied to it; L_2 is called the *category* or *class representation* layer because it is the layer where category representations, i.e., the clusters to which input patterns belong, are formed. The orienting subsystem consists of a single node (the *reset* node), which accepts inputs from the nodes in L_1 and L_2 and the input pattern directly; its output affects the nodes in L_2.

Patterns are assumed to be n-dimensional vectors in $[0,1]^n$; the input to the Fuzzy ART network is formed by putting the pattern and its complement in a $2n$-dimensional vector $\mathbf{x}$.

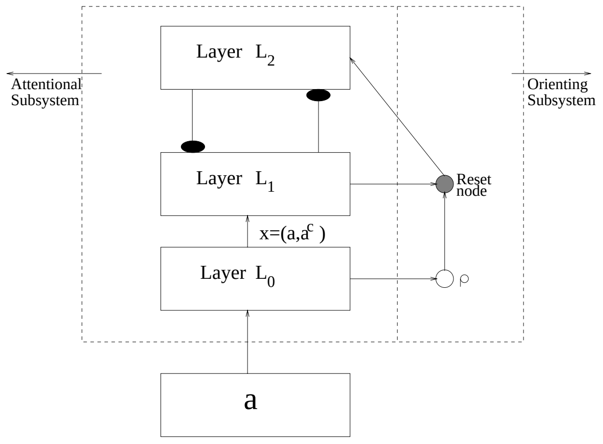

Figure 6.11 Network architecture of a Fuzzy ART.

Each category j in L_2 corresponds to a vector $\mathbf{w}_j = (w_{j1}, \dots, w_{j,2n})$ of weights. All these weights are initially set to one: such a category is said to be *uncommitted*. After a category has been chosen to represent an input pattern, it is referred to as a *committed* category or node.

Training a Fuzzy ART means tuning its weights so as to cluster the input patterns $\mathbf{x}_1, \dots, \mathbf{x}_P$ into different categories, according to their similarity. The dataset of patterns is repeatedly presented to the Fuzzy ART in the given order, as many times as necessary. Training is considered accomplished when the weights do not change during a complete dataset presentation. This training scenario is called *off-line* training.

Off-line training proceeds as follows: each time an input pattern $\mathbf{x}_i$ is presented, the input $T_j(\mathbf{x}_i)$ to each category node j is calculated as

$$T_j(\mathbf{x}_i) = \begin{cases} \frac{n}{\alpha+2n}, & \text{if } j \text{ is uncommitted;} \\ \frac{|\mathbf{x}_i \wedge \mathbf{w}_j|}{\alpha+|\mathbf{w}_j|}, & \text{if } j \text{ is committed.} \end{cases} \tag{6.28}$$

Now, let's call j^* the node in L_2 receiving the maximum input from L_1,

$$j^* = \arg\max_{j \in L_2} T_j(\mathbf{x}_i).$$

Two cases require special actions:

1. if node j^* is uncommitted, a new uncommitted node in L_2 is introduced, and its weights are all initialized to one;

2. if node j^* is committed but
$$\frac{|\mathbf{x}_i \wedge \mathbf{w}_j|}{|\mathbf{x}_i|} < \rho,$$
(i.e., it does not satisfy the vigilance criterion), it is disqualified by setting $T_{j^*}(\mathbf{x}_i) = -1$ and the next maximum-input node in L_2 is considered, until either case 1 is verified or j^* satisfies the vigilance criterion.

At this point, the weights associated with node j^* are modified according to the equation
$$\mathbf{w}_{j^*} \leftarrow \mathbf{w}_{j^*} \wedge \mathbf{x}_i.$$

Quantity $\rho \in [0, 1]$ is the *vigilance* parameter, affecting the resolution of clustering: small values of ρ result in coarse clustering, whereas larger values result in finer clustering of input patterns. Parameter $\alpha \in (0, +\infty)$ is called the *choice* parameter.

The $\wedge$ operation in the above equations is the componentwise minimum, and could be replaced in principle by any t-norm.

Fuzzy ARTMAP

A Fuzzy ART module generates the categories needed to classify the input by unsupervised learning, according to a similarity criterion. A composition of Fuzzy ART modules makes up a Fuzzy ARTMAP [40], a neuro-fuzzy architecture capable of learning the relationship between data and user-defined categories (supervised learning).

A Fuzzy ARTMAP, illustrated in Figure 6.12, consists of two Fuzzy ART modules, ART$_a$ and ART$_b$, plus a MAP module. ART$_a$ receives in input the patterns to be classified; ART$_b$ receives in input the m user-defined classes to which the patterns belong, and generates an internal category layer corresponding to it. The MAP module connects the two Fuzzy ART modules and tries to minimize the difference between the classes generated by ART$_a$ from data and the user-defined classes generated by ART$_b$. In other words, the MAP module builds a mapping from the n inputs to the m classes, possibly by acting on the vigilance parameter ρ_a of ART$_a$.

Among the main features of Fuzzy ART and Fuzzy ARTMAP are the minimization of the predictive error, the maximization of code compression and generalization, the dynamic introduction of new categories when needed, the possibility of distinguishing exceptions from noise, and fast convergence. All these features make this type of neuro-fuzzy architecture very popular in a variety of applications.

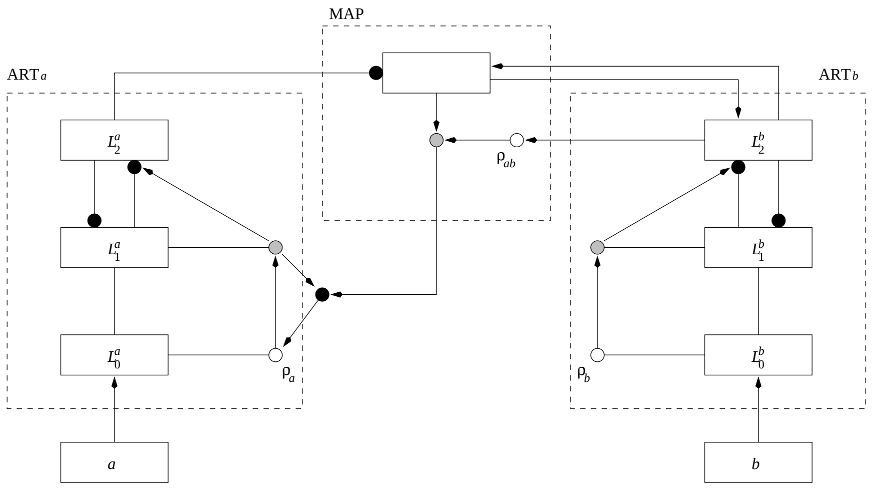

Figure 6.12 Network architecture of a Fuzzy ARTMAP.

6.4 Applications of Neuro-fuzzy Systems

Neuro-fuzzy systems are passing from their infancy, where research on foundations and methodologies predominates over applications, to maturity, where principles and methodologies become technology deployed on the field. This passage has been fostered by several methodological developments that have been described in the previous sections. In this last section, which roughly follows Chapter 14 of [223], we give an overview of neuro-fuzzy systems applications that have gained their acceptance on the field, without any ambition of being comprehensive. In fact, new valuable applications appear every year, and any attempt at making a survey would soon become obsolete.

6.4.1 Engineering

Neuro-fuzzy approaches have been used in a variety of engineering applications, including consumer electronics, control, diagnostics, manufacture, biotechnology, power generation, chemical processes, power electronics, communications, and software resource management. It is now rather well established that neuro-fuzzy systems can adequately adapt to changing environmental conditions.

Ishibuchi and colleagues [102] applied an ANFIS-like fuzzy neural network (Sugeno-type fuzzy rules with gradient descent learning) to rice tasting, which involves the development of a six-variable fuzzy relation.

Of particular interest to our synergetic vision of soft computing is the work reported by Pao [162], where neural networks, fuzzy logic, and evolutionary algorithms are combined to support the task of process monitoring and optimization in electric power utilites, including heat rate improvement and NO emission minimization.

6.4.2 Diagnostics in Complex Systems

Neuro-fuzzy systems have been applied to several problems arising in the aerospace industry, like control surface failure detection for a high-performance aircraft [184]. The detection model is developed using a linear 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 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 neuro-fuzzy decision algorithm. Simulation results demonstrate that detection can be achieved without false alarms even in the presence of actuator/sensor dynamics and noise.

Typical examples of complex systems whose modeling and monitoring is of critical importance are nuclear reactors. Several applications of neuro-fuzzy techniques have been described in the literature [224].

6.4.3 Control

Neuro-fuzzy systems have found broad application in control, probably more so than in any other field. Controlled plants are as diverse as industrial sewing machines [213] and fusion reactors [250], home electric appliances [236] like refrigerators, air-conditioning systems, and welding machines, and consumer electronic devices such as handheld video cameras.

6.4.4 Robotics

In the field or robotics, neuro-fuzzy systems have been employed for supervisory control, planning, grasping, and guidance. Grasping

[61] has to do with the control of robotic arms with three or more fingers; the main issue is finding an optimal coordination of the forces applied to the object, in order to hold it firmly without squeezing or damaging it. On-line learning ensures that grasp parameters are continuously adjusted to current conditions.

Approaches based on Fuzzy ART have been used for autonomous robot guidance and navigation: for instance, Bonarini and Ludovico [30] report on a system that is able to learn a topological map derived from different robot sensors, and use it to navigate in unstructured environments. Their approach is based on the definition and identification of grounded concepts by integrated instances of Fuzzy ART.

6.4.5 Image Processing

Image processing and pattern recognition are a field in which neural networks play a prominent role, especially because it naturally lends itself to massively parallel and connectionist processing of the type supported by ANNs. However, in recent years, neuro-fuzzy systems have been designed and investigated to improve on ANN performance in image processing tasks both at low level, such as in image quality enhancement and image manipulation, and high level, such as in edge detection, pattern recognition, medical and environmental imaging.

6.4.6 Finance

ANNs have been used for years by the financial community in investment and trading because of their ability to identify patterns of behavior that are not readily available and complex relations among variables that are hidden by chaos and noise. Much of this work has never been published for obvious reasons, although generic and casual accounts of it have been given.

Since the mid 1990s, neuro-fuzzy techniques have been incorporated into some financial applications. For instance, a neuro-fuzzy decision support system is described in [96], which tries to determine whether a given stock is underpriced or overpriced by learning patterns associated with either condition.

Another neuro-fuzzy system [39] is capable of evaluating portfolios by using currency exchange rate fluctuations and expert knowledge given in the form of fuzzy IF-THEN rules; the merit of the neuro-fuzzy system is its ability to automatically fine tune the expert knowledge with a backpropagation-like algorithm.