15.7 Integration Example: Subsethood-Product Based Fuzzy–Neural Inference System

15.7.1 Description of the Model

The Subsethood Product Fuzzy–Neural Inference System (SuPFuNIS) [445–447] uses a standard fuzzy–neural network architecture that embeds fuzzy *if–then* rules into a network architecture. In fig. 15.5 portrays this architecture. Figure 15.5, input nodes denote features, output nodes denote

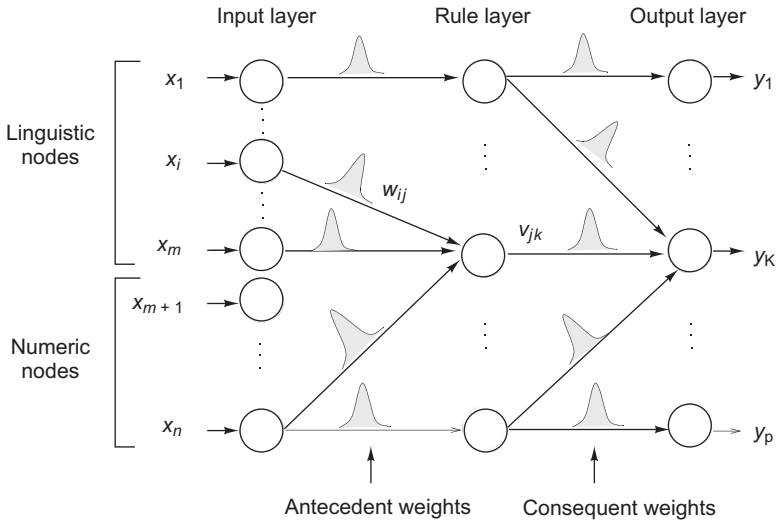

Fig. 15.5 Architecture of the SuPFuNIS model [446]. (©2002 IEEE. Reprinted with permission)

A number of earlier variants of the SuPFuNIS model with applications in function approximation, inference and classification have been presented elsewhere [444, 447, 448]. In [448] a combination of weighted subsethood and soft-minimum conjunction operator was employed. The model used a triangular approximation instead of Gaussian fuzzy weights for subsethood computation. It addressed the applications of function approximation and inference. In [444], which extended [448] by increasing the number of free parameters, a simple heuristic to derive the number of rules using clustering was introduced. A combination of mutual subsethood and product conjunction operator with a non-tunable feature fuzzifier has been presented in [447]. The network in [447] uses Gaussian fuzzy weights and targets the classification problem domain.

classes, and each hidden node represents a rule. Fuzzy rule antecedents translate to input-hidden node connections, and fuzzy rule consequents translate to hidden-output node connections. Fuzzy sets corresponding to linguistic labels of these fuzzy *if–then* rules are represented by symmetric Gaussian membership functions, identified by a center and spread. Therefore a fuzzy weight w_{ij} from input node i to rule node j is modelled by the center w_{ij}^c and spread w_{ij}^σ of a Gaussian fuzzy set: $w_{ij} = (w_{ij}^c, w_{ij}^\sigma)$. Similarly, the consequent fuzzy weight from a rule node j to output node k is denoted by $v_{ij} = (v_{ij}^c, v_{ij}^\sigma)$.

The novelty of this model lies in its simultaneous admission of numeric as well as fuzzy inputs. Numeric inputs are first fuzzified so that all inputs to the network are uniformly fuzzy. Since the antecedent weights are also fuzzy, in SuPFuNIS signal transmission along the fuzzy weight is based on a mutual subsethood measure.

Signal Transmission at Input Nodes

An input feature x_i can be either numeric or linguistic, and therefore there are two kinds of nodes in the input layer (see Fig. 15.5). A linguistic node accepts an input in the form of a fuzzy set with a Gaussian membership function defined by a center x_i^c and spread x_i^σ. The signal transmitted out of a linguistic node is of the form $s_i = (x_i^c, x_i^\sigma)$ since no transformation of inputs takes place at these nodes.

Numeric nodes are feature-specific fuzzifiers. A numeric node accepts a numeric input x_i and fuzzifies it into a fuzzy set by treating x_i as the center of a Gaussian membership function with a spread x_i^σ. Once again the node transmits a signal of the form $s_i = (x_i^c, x_i^\sigma)$. Fuzzy signals are transmitted to hidden rule nodes through fuzzy weights w_{ij}.

Mutual Subsethood Based Signal Transmission

Since both the signal and the weight are fuzzy sets, being represented by Gaussian membership functions, the net value of the signal transmitted along the weight is quantified by the extent of overlap between the two fuzzy sets. This is measured by their *mutual subsethood* which is defined for two fuzzy sets A, B as

$$\mathcal{E}(A, B) = \frac{\mathcal{C}(A \cap B)}{\mathcal{C}(A) + C(B) - \mathcal{C}(A \cap B)} \tag{15.1}$$

The cardinality of a fuzzy set A is defined as $\mathcal{C}(A) = \int_{-\infty}^{\infty} e^{-((x-c)/\sigma)^2} dx$.

where $\mathcal{C}(\cdot)$ is the fuzzy set cardinality. The mutual subsethood measure can have values in the interval (0,1], and depends upon the relative values of centers and spreads of fuzzy sets.

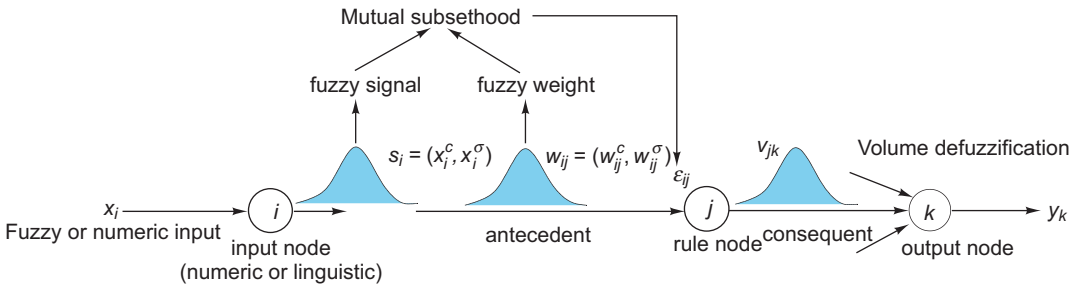

Fig. 15.6 Fuzzy signal transmission in SuPFuNIS (modified from [446])

As shown in Fig. 15.6, in SuPFuNIS, a fuzzy input signal is transmitted along a fuzzy weight, and the net contribution of that input to rule node j is quantified by $\mathcal{E}_{ij}$, the mutual subsethood between the fuzzy signal

$s_i = (x_i^c, x_i^\sigma)$ and fuzzy weight $w_{ij} = (w_{ij}^c, w_{ij}^\sigma)$:

$$\begin{aligned}\mathcal{E}_{ij} &= \mathcal{E}(s_i, w_{ij}) \\ &= \frac{\mathcal{C}(s_i \cap w_{ij})}{\mathcal{C}(s_i) + \mathcal{C}(w_{ij}) - \mathcal{C}(s_i \cap w_{ij})}\end{aligned} \tag{15.2}$$

Various different cases of overlap can arise. As an example, consider the case when $x_i^c = w_{ij}^c$. If $x_i^\sigma < w_{ij}^\sigma$ then the signal fuzzy set s_i completely belongs to the weight fuzzy set w_{ij} and the cardinality $\mathcal{C}(s_i \cap w_{ij}) = \mathcal{C}(s_i)$. This can be evaluated as follows:

$$\begin{aligned}\mathcal{C}(s_i \cap w_{ij}) = \mathcal{C}(s_i) &= \int_{-\infty}^{\infty} e^{-(x - x_i^c/x_i^\sigma)^2}\, dx \\ &= x_i^\sigma \sqrt{\pi}\end{aligned} \tag{15.3}$$

Similarly, $\mathcal{C}(s_i \cap w_{ij}) = \mathcal{C}(w_{ij})$ if $x_i^\sigma > w_{ij}^\sigma$ and $\mathcal{C}(s_i \cap w_{ij}) = w_{ij}^\sigma \sqrt{\pi}$. If $x_i^\sigma = w_{ij}^\sigma$, the two fuzzy sets are identical. In summary, for $x_i^c = w_{ij}^c$,

$$\mathcal{C}(s_i \cap w_{ij}) = \begin{cases} \mathcal{C}(s_i) = x_i^\sigma \sqrt{\pi} & \text{if } x_i^\sigma < w_{ij}^\sigma \\ \mathcal{C}(w_{ij}) = w_{ij}^\sigma \sqrt{\pi} & \text{if } x_i^\sigma > w_{ij}^\sigma \\ \mathcal{C}(s_i) = \mathcal{C}(w_{ij}) = x_i^\sigma \sqrt{\pi} = w_{ij}^\sigma \sqrt{\pi} & \text{if } x_i^\sigma = w_{ij}^\sigma \end{cases} \tag{15.4}$$

Details of other cases are given in [446]. The corresponding expressions for $\mathcal{E}(s_i, w_{ij})$ are obtained by substituting for $\mathcal{C}(s_i \cap w_{ij})$ into Eq. 15.2.

Activity Aggregation at Rule Nodes

By measuring the value of mutual subsethood $\mathcal{E}_{ij}$ for a rule node j we are in essence assessing the compatibility between the signal s_i and the fuzzy weight w_{ij}. Each rule node is expected to somehow aggregate the vector $\mathbf{E}_j = (\mathcal{E}_{1j}, \ldots, \mathcal{E}_{nj})$ in such a way that the resulting node activation reflects the extent to which that rule fires. The activation z_j, of rule node j is a mutual subsethood based product:

$$z_j = \prod_{i=1}^{n} \mathcal{E}_{ij} = \prod_{i=1}^{n} \mathcal{E}(s_i, w_{ij}) \tag{15.5}$$

No other transformation of z_j occurs at a rule node and this numeric activation value is transmitted unchanged to consequent connections.

Output Layer Signal Computation

The signal of each output node is determined using standard volume based centroid defuzzification [325]. Since the area of consequent weights (represented by Gaussian fuzzy sets) is $v_{jk}^\sigma \sqrt{\pi}$, then,

$$y_k = \frac{\sum_{j=1}^{q} z_j v_{jk}^c v_{jk}^\sigma}{\sum_{j=1}^{q} z_j v_{jk}^\sigma} \tag{15.6}$$

which is the numeric output obtained. The defuzzifier (Eq. 15.6) essentially computes a convex sum of consequent set centers. This completes our discussion on how input vectors are mapped to outputs in SuPFuNIS.

Supervised Learning

The SuPFuNIS is trained using standard gradient descent learning. This involves repeated presentation of a set of input patterns drawn from the training set. The output of the network is compared with the desired value to obtain the error, and network weights are changed on the basis of a square error minimization criterion. Once the network is trained to the desired level of error, it is tested by presenting a new set of input patterns drawn from the test set. The squared error E_k used as a training performance parameter is:

$$E_k = \frac{1}{2}\sum_{j=1}^{p}(d_j^k - y_j^k)^2 \tag{15.7}$$

where d_j^k is the desired value at output node j on iteration k, and the error is evaluated over all p outputs for a specific pattern k. For a 1-of-C class classification the desired outputs will be 0 or 1. Both the centers and spreads w_{ij}^c, v_{jk}^c, w_{ij}^σ, v_{jk}^σ, of antecedent and consequent connections, and the spreads of the input features x_i^σ are modified on the basis of iterative update equations [446].

15.7.2 Application: Truck Backer-upper Control Problem

The SuPFuNIS model finds application in a variety of domains. These include function and time series approximation; data classification; diagnosis; and control [446]. Here we describe a control application.

The suitability of SuPFuNIS for control applications is demonstrated on the truck backer-upper problem which deals with backing up a truck to a loading dock. The truck corresponds to the cab part of the truck in the Nguyen–Widrow [412] neural truck backer-upper system. The truck position is exactly determined by three state variables ϕ, x, and y, where ϕ is facilitated through the angle of the truck with the horizontal, x and y are the coordinates in the space as depicted in Fig. 15.7. The control of the truck is facilitated through the steering angle θ. The truck moves backward by a fixed unit distance every stage. We also assume enough clearance between the truck and the loading dock such that the coordinate y does not have to be considered as an input. We design a control system, whose inputs are $\phi \in [-90^o, 270^o]$ and $x \in [0, 20]$, and whose output is $\theta \in [-40^o, 40^o]$, such that the final state will be $(x_f, \phi_f) = (10, 90^o)$.

For a validation of this assumption refer to [319].

The following kinematic equations are used to simulate the control system [581]:

$$x(t+1) = x(t) + \cos(\phi(t) + \theta(t)) + \sin(\theta(t))\sin(\phi(t)) \tag{15.8}$$

$$y(t+1) = y(t) + \sin(\phi(t) + \theta(t)) - \sin(\theta(t))\cos(\phi(t)) \tag{15.9}$$

$$\phi(t+1) = \phi(t) - \sin^{-1}\left(\frac{2\sin(\theta(t))}{b}\right) \tag{15.10}$$

where b is the length of the truck and is assumed as 4 for the present

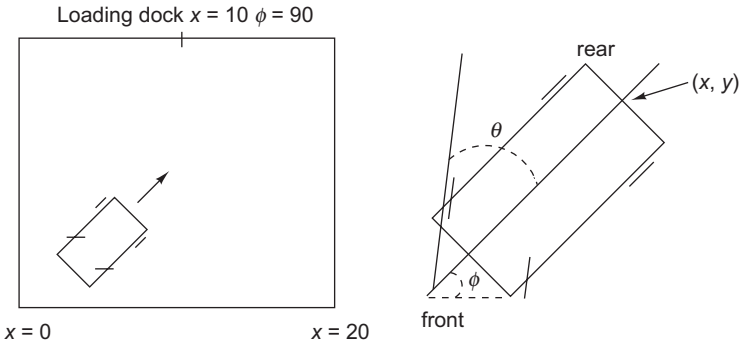

Fig. 15.7 Diagram of simulated truck and loading zone [446]
(©2002 IEEE. Reprinted with permission)

simulation.

As mentioned above, the training data (adapted from [581]) comprise 238 pairs which are accumulated from 14 sequences of desired $(x, \phi; \theta)$ values. The 14 initial states (x_0, ϕ_0) are (1,0), (1,90), (1,270), (7,0), (7,90), (7,180), (7,270), (13,0), (13,90), (13,180), (13,270), (19,90), (19,180), (19,270). Three initial states, (x_0, ϕ_0) = (3, –30), (10,220), and (13,30) were used to test the performance of the controller. The number of trainable parameters for this application is $(6r+2)$ for an r rule SuPFuNIS network.

We used a normalized variant of the *docking error* (which essentially measures the Euclidean distance from the actual final position (ϕ_a, x_a) to the desired final position (ϕ_f, x_f)), as well as the *trajectory error* (the ratio of the actual length of the trajectory to the straight line distance from the initial point to the loading dock) as performance measures (derived from [319]):

$$\text{Normalized Docking Error} = \sqrt{\left(\frac{\phi_f - \phi_a}{360}\right)^2 + \left(\frac{x_f - x_a}{20}\right)^2} \quad (15.11)$$

$$\text{Trajectory Error} = \frac{\text{length of truck trajectory}}{\text{distance (initial position, desired final position)}} \quad (15.12)$$

The docking errors for three test points for three rules and five rules are shown in Table 15.3. These results demonstrate that SuPFuNIS is able to perform very well—with a high docking accuracy—with just 5 rules. The simulation results are to be compared with Kosko and Kong's fuzzy controller for backing up the truck to the dock that used 35 linguistic rules [319], and the Wang–Mendel controller [581] that used 27 rules which are either linguistic or a mixture of linguistic and rules obtained from numeric data. The truck trajectories from three initial states are shown in Fig. 15.8.

The SuPFuNIS is able to successfully generate low error trajectories from each of the initial test points. In addition in SuPFuNIS one can incorporate

Table 15.1 Docking errors for numeric data (adapted from [446])

Initial point (x, y, ϕ^o)	Rules	Normalized Docking Error	Trajectory Error
(3,3,–30)	3	0.0087	1.2116
(10,4,220)	3	0.0119	1.2737
(13,3,30)	3	0.0073	1.0540
(3,3,–30)	5	0.0088	1.2106
(10,4,220)	5	0.0120	1.2059
(13,3,30)	5	0.0058	1.0533

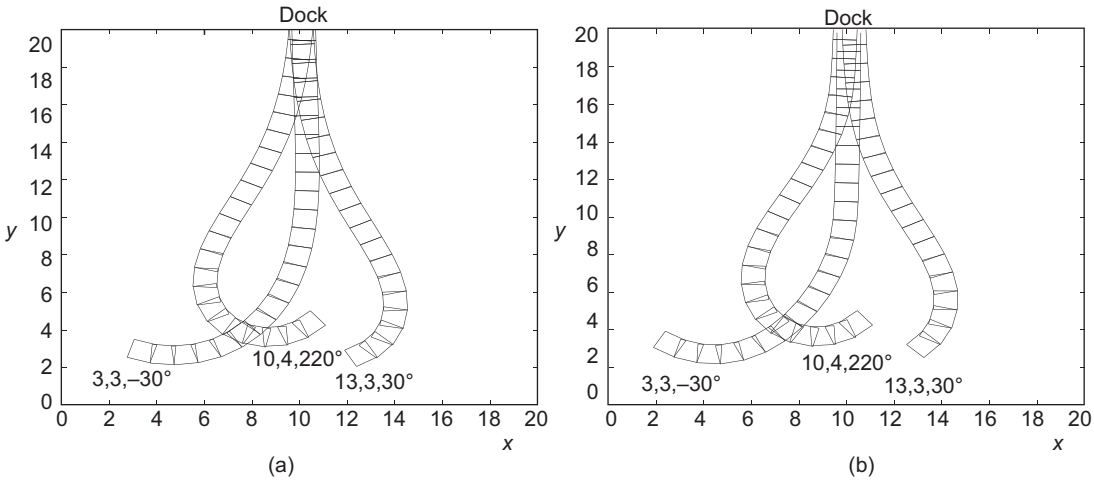

Fig. 15.8 Truck trajectories from three testing points $(3, 3, -30^o)$, $(10, 4, 220^o)$, $(13, 3, 30^o)$ (a) using 3 rules and (b) using 5 rules [446]
(©2002 IEEE. Reprinted with permission)

expert knowledge easily and seamlessly into the network. For more details, the reader is referred to [446].

15.7.3 Evolvable SuPFuNIS

Genetic Learning

In a real-coded genetic algorithm the basic operation is similar to the binary GA described in the text with the difference arising in the nature of the crossover and mutation operators.

To overcome the problems commonly encountered with gradient descent algorithms, genetic learning was introduced into the SuPFuNIS model. The resulting model is called the Evolvable Subsethood-Product Fuzzy–Neural Inference System (ESuPFuNIS) [564]. The ESuPFuNIS

employs a real-coded genetic algorithm (RGA) to evolve all the trainable parameters of the network which includes the input feature spreads, and the antecedent and consequent fuzzy weights. These correspond to the variables: x_i^σ, w_{ij}^c, v_{jk}^c, w_{ij}^σ, v_{jk}^σ.

For an n-q-p network (n input features, q rule (hidden) nodes and p output nodes) the total number of evolvable parameters is $(n + 2q(n + p))$. There are n inputs and q rule nodes which give $2nq$ parameters. Add to this n input spreads, one for each feature. On the consequent side there are $2qp$ free parameters. This gives $(n + 2q(n + p))$.

Genetic Coding

We first describe the coding scheme employed. Figure 15.9 shows this genetic coding scheme for the parameters used in the RGA. Each chromosome comprises three parts. The first part involves feature spreads associated with the input fuzzifier. The second part involves antecedent fuzzy weights. The

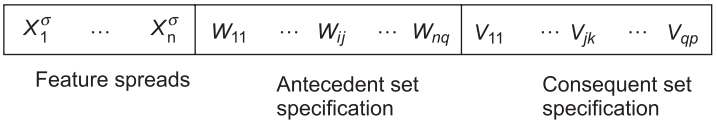

Fig. 15.9 String representation of feature spreads and fuzzy weights for the RGA

antecedent fuzzy weights of all the connections that fan in to each rule node are coded contiguously. The third and last part concatenates all consequent fuzzy weights that fan-in to each of the output nodes.

Genetic Algorithm Operators and Operations

We mentioned earlier that the nature of the operators employed in the RGA are rather different from those employed in the simple GA. We describe one set of them here—those that were used for optimization on ESuPFuNIS. The implementation of the RGA in ESuPFuNIS employs specialized selection, crossover, and mutation operators, as wells as a novel procedure to generate the new population.

Crossover The *blend crossover* operator (BLX-α) [140] is employed in ESuPFuNIS for performing the crossover operation. Let C_j^1, C_j^2 represent the jth gene of two parent chromosomes C^1 and C^2 each of string size s. If $C_{\max} = \max(C_j^1, C_j^2)$, and $C_{\min} = \min(C_j^1, C_j^2)$, then we define $I = C_{\max} - C_{\min}$, $j = 1, \cdots, s$. BLX-α generates a single offspring $H = (h_1, \cdots, h_j, \cdots, h_s)$ where h_j is a randomly chosen number from the interval $[C_{\min} - I \cdot \alpha, C_{\max} + I \cdot \alpha]$.

In BLX-α, the value of α determines the two-sided extension of interval I from which the crossover gene value is ultimately selected at random.

Mutation The ESuPFuNIS employs a *non-uniform mutation* (NUM) operator [388] in its RGA. If T is the maximum number of generations, then NUM, when applied to a generation t, mutates a gene $C_i \in [a_i, b_i]$ to C_i' as follows:

$$C_i' = \begin{cases} C_i + \Delta(t, b_i - C_i) & \text{if } \tau = 0 \\ C_i - \Delta(t, C_i - a_i) & \text{if } \tau = 1 \end{cases} \tag{15.13}$$

Here, $\tau \in \{0, 1\}$, $\Delta(t, y) = y\left(1 - r^{(1-\frac{t}{T})^b}\right)$, r is a random number from the interval $[0, 1]$, and b is a user defined parameter that decides the degree of

The NUM operator function $\Delta(t, y) = y\left(1 - r^{(1-\frac{t}{T})^b}\right)$ is designed to generate perturbation values on the range $[0, y]$ in such a way that the probability of the function returning numbers close to zero increases as the algorithm steps elapse.

dependency on the number of generations.

The selection mechanism essentially determines the diversity of the population, which depends upon the *selection pressure*. The selection pressure is the degree to which the selection mechanism favours better individuals. In the present case, the selection pressure increases with a decrease in $\eta_{\min}$.

Selection The RGA in ESuPFuNIS employs a *linear ranking selection* and *stochastic universal sampling* technique. Linear ranking selection ranks chromosomes according to their fitness values with the best and worst chromosomes carrying ranks 1 and N respectively, where N is the population size. The selection probability $p_s(C)$ of each chromosome C is calculated according to its rank as follows:

$$p_s(C) = \frac{1}{N}\left(\eta_{\max} - (\eta_{\max} - \eta_{\min})\frac{\text{rank}(C^i) - 1}{N - 1}\right) \tag{15.14}$$

Here, C^i is the ith chromosome with $i = 1, \cdots, N$; $\eta_{\min}$ and $\eta_{\max}$ denote the expected number of copies for the worst and the best chromosomes respectively with $\eta_{\min} \in [0, 1]$ and $\eta_{\max} = 2 - \eta_{\min}$. Stochastic universal sampling by simulating a roulette wheel which selects chromosomes for the next generation based on the selection probabilities.

The values of various RGA parameters employed by ESuPFuNIS were: $N = 100$, $p_c = 0.6$, $p_m = 0.1$, $\alpha = 0.5$, $\eta_{\min} = 0.5$, $b = 5$. More details on the implementation algorithm are to be found in [564].

15.7.4 Application: Ripley's Synthetic Data Classification

The Ripley synthetic data set is available from `markov.stats.ox.ac.uk/pub/PRNN`. It comprises two dimensional patterns belonging to two classes. Each class has a bimodal distribution generated from equal mixtures of Gaussian distributions with identical covariance matrices [477]. The class distributions have been chosen to allow a Bayesian classifier error rate of 8 per cent. The training set consists of 250 patterns with 125 patterns in each class. The test set consists of 1000 patterns with 500 patterns in each class.

For this classification problem, ESuPFuNIS employed a 2-2-2 architecture. A real coded genetic algorithm was employed to evolve the network parameters. The fitness function combines the root mean squared error with the misclassification error as follows:

$$F = \sqrt{\frac{1}{Q}\sum_{k=1}^{Q}\left(\frac{1}{p}\sum_{o=1}^{p}(d_o^k - y_o^k)^2\right)} + \frac{1}{Q}\sum_{k=1}^{Q}(c_k \neq \hat{c}_k) \tag{15.15}$$

where Q is the number of learning patterns involved; p is the number of outputs; d_o^k and y_o^k are the desired and the actual oth outputs of the network for the kth learning pattern and $\hat{c}_k$ and c_k are the desired and the actual classification of the kth learning pattern by the network.

The ESuPFuNIS gives a test error rate of 7.6 per cent, a performance which is better than that of the Bayes classifier. The class separating boundary learnt by ESuPFuNIS is shown in Fig. 15.10. The test error rate of 7.6 per cent of ESuPFuNIS with two rules, marks an improvement over

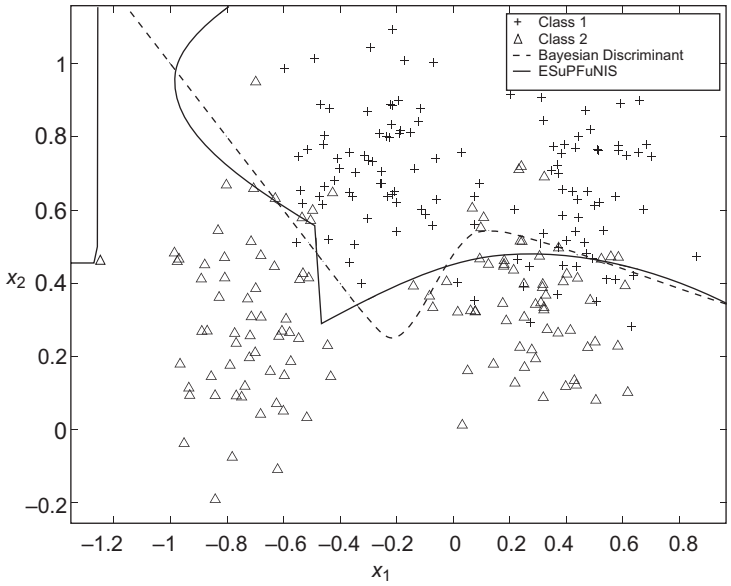

Fig. 15.10 Class separating boundary learnt by ESuPFuNIS (7.6 per cent error) along with the Bayesian (8.0 per cent error) decision boundary for Ripley's training data

SuPFuNIS which was able to achieve the same error with 10 rules, both in terms of classification accuracy and rule economy. It also performs better than various other models as presented in [564].