6

Neuro-fuzzy Approach

6.1 Motivation for Technology Merging

Contemporary intelligent technologies have various characteristic features that can be used to implement systems that mimic the behaviour of human beings. For example, expert systems are capable of reasoning about the facts and situations using the rules out of a specific domain, *etc*. The outstanding feature of neural networks is their capability of learning, which can help in building artificial systems for pattern recognition, classification, *etc*. Fuzzy logic systems, again, are capable of interpreting the imprecise data that can be helpful in making possible decisions. On the other hand, genetic algorithms provide implementation of random, parallel solution search procedures within a large search space. Therefore, in fact, the complementary features of individual categories of intelligent technologies make them ideal for isolated use in solving some specific problems, but not well suited for solving other kinds of intelligent problem. For example, the black-box modelling approach through neural networks is evidently well suited for process modelling or for intelligent control, but less suitable for decision making. On the other hand, the fuzzy logic systems can easily handle imprecise data, and explain their decisions in the context of the available facts in linguistic form; however, they cannot automatically acquire the linguistic rules to make those decisions. Such capabilities and restrictions of individual intelligent technologies have actually been a central driving force behind their fusion for creation of ***hybrid intelligent systems*** capable of solving many complex problems.

The permanent growing interest in intelligent technology merging, particularly in merging of neural and fuzzy technology, the two technologies that complement each other (Bezdek, 1993), to create neuro-fuzzy or fuzzy-neural structures, has largely extended the capabilities of both technologies in hybrid intelligent systems. The advantages of neural networks in learning and adaptation and those of fuzzy logic systems in dealing with the issues of human-like reasoning on a linguistic level, transparency and interpretability of the generated model, and handling of uncertain or imprecise data, enable building of higher level intelligent systems. The

synergism of integrating neural networks with fuzzy logic technology into a hybrid functional system with low-level learning and high-level reasoning transforms the burden of the tedious design problems of the fuzzy logic decision systems to the learning of connectionist neural networks. In this way the approximation capability and the overall performance of the resulting system are enhanced.

A number of different schemes and architectures of this hybrid system have been proposed, such as ***fuzzy-logic-based neurons*** (Pedrycz, 1995), ***fuzzy neurons*** (Gupta, 1994), ***neural networks with fuzzy weights*** (Buckley and Hayashi, 1994), ***neuro-fuzzy adaptive models*** (Brown and Harris, 1994), *etc*. The proposed architectures have been successful in solving various engineering and real-world problems, such as in applications like system identification and modelling, process control, systems diagnosis, cognitive simulation, classification, pattern recognition, image processing, engineering design, financial trading, signal processing, time series prediction and forecasting, *etc*.

6.2 Neuro-fuzzy Modelling

There are several methods for implementing the neuro-fuzzy modelling technique. An early merging approach was to replace the input-output signals or the weights in neural networks by membership values of fuzzy sets, along with the application of fuzzy neurons (Mitra and Hayashi, 2000). Several authors have proposed an internal structure for fuzzy neurons (Gupta, 1994; Buckley and Hayashi, 1995), as presented in the following section.

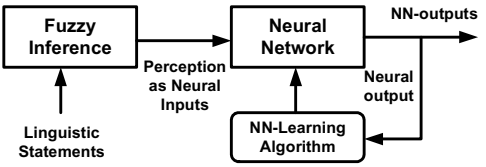

Figure 6.1. (a) Fuzzy-neural system (first model)

In general, neuro-fuzzy hybridization is done in two ways (Mitra and Hayashi, 2000):

- a neural network equipped with the capability of handling fuzzy information processing, termed a fuzzy-neural network (FNN)
- a fuzzy system augmented by neural networks to enhance some of its characteristics, like flexibility, speed, and adaptability, termed a neural-fuzzy system (NFS).

Neural networks with fuzzy neurons are also termed FNN, because they are also capable of processing fuzzy information. A neural-fuzzy system (NFS), on the other hand, is designed to realize the process of fuzzy reasoning, where the

connection weights of the network correspond to the parameters of fuzzy reasoning (Nauck *et al*., 1997).

Gupta (1994) has presented two additional models for fuzzy neural systems. The first model (Figure 6.1(a)) consists of a fuzzy inference block, followed by a neural network block, consisting of a multilayer feedforward neural network, the input of which is fed by the inference block (Fuller, 1995). The neural network used can be adapted and adequately trained with training samples to yield the desired outputs.

In the second model (Figure 6.1(b)), the neural network block drives the fuzzy inference system to generate the corresponding decisions. Hence, the first model takes linguistic inputs and generates the numerical outputs, whereas the second model takes numerical inputs and generates the linguistic outputs.

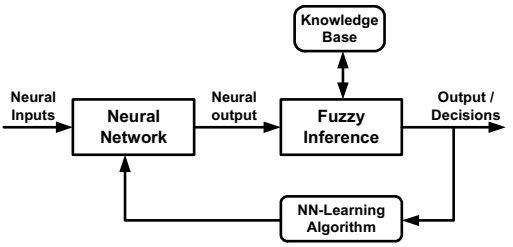

Figure 6.1. (b) Fuzzy-neural system (second model)

Alternatively, the second approach is to use fuzzy membership functions to pre-process or post-process signals with neural networks as shown in Figure 6.2. A fuzzy inference system can encode an expert's knowledge directly and easily using rules with linguistic labels (Kulkarni, 2001).

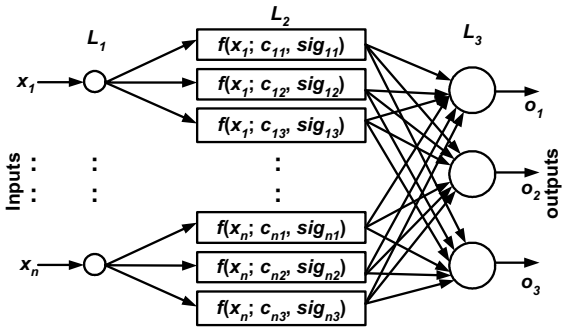

Figure 6.2. Fuzzy-neural model with tuneable membership function

In practice, for optimal tuning of membership functions of the fuzzy logic part of a neuro-fuzzy system, a reliable skill is required. The incorporated neural network part of the same system can, using its learning capability, perform on-line

tuning of membership functions and gradually improve the performance of the entire hybrid system. This concept, which became very popular in engineering applications, was originally proposed and extended to multidimensional membership functions by Takagi and Hayashi (1991).

Lin and Lee (1991) proposed a neural-network-based model for fuzzy logic control consisting of a feedforward neural network, the input nodes of which are fed by input signals and its output nodes delivering the output and decision signals. Nodes in the hidden layers of the system implement the membership functions and the fuzzy rules, making up a fuzzy inference system with distributed representation and learning algorithms of the neural network. Parameters representing membership functions are determined using any suitable network training algorithm. Pal and Mitra (1992) proposed a similar model in which inputs are fed to a preprocessor block, which performs the same functions as that in the above fuzzy inference system. The output of the preprocessor delivers the fuzzy membership function values. For each input variable term, linguistic labels such as *low*, *medium*, and *high* are used. If input consists of n variables, then the preprocessor block yields $m \times n$ outputs, where m represents the number of term values used in the model. The output of the preprocessor block is then fed to a multilayer perceptron model that implements the inference engine. The model was successfully used for classifying vowels in English alphabets. Kulkarni (1998), again, developed a similar model and successfully used it for multi-spectral image analysis. Some authors have designed neuro-fuzzy systems incorporating some processing stages implemented with neural networks and some with a fuzzy inference system. In another design, a neural-network-based tree classifier was used. Finally, Kosko (1992) suggested some remarkable neuro-fuzzy models for fuzzy associative memory (FAM).

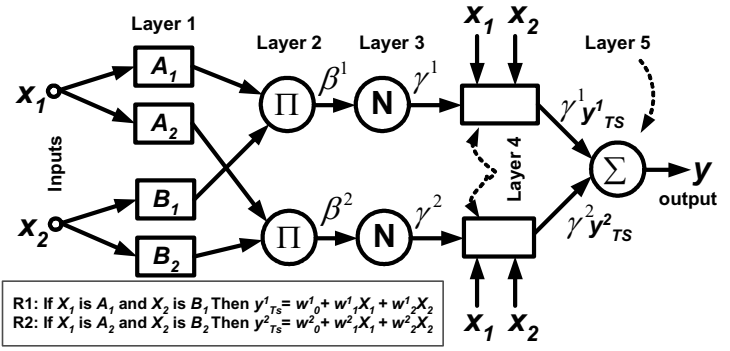

Figure 6.3. ANFIS architecture with Takagi-Sugeno-type fuzzy model with two rules

The neuro-fuzzy model ANFIS (adaptive-network-based fuzzy inference system) of Jang (1993), presented in Figure 6.3, incorporates a five-layer network to implement a Takagi-Sugeno-type fuzzy system. The proposed model has a relatively complex architecture for a large number of inputs, and it can process a large number of fuzzy rules. It uses the least mean square training algorithm in the

forward computation to determine the linear consequents of the Takagi-Sugeno rules, while for the optimal tuning of an antecedent membership function backpropagation is used (Kim and Kim, 1997).

The neuro-fuzzy model of Chak *et al.* (1998) can locate the fuzzy rules and optimize their membership functions by competitive learning and a Kalman filter algorithm. The key feature is that a high-dimensional fuzzy system can be implemented with fewer rules than that required by a conventional Sugeno-type model. This is because the input space partitions are unevenly distributed, thus enabling a real-time network implementation.

The approach of Nie (1997) concerns the development of a ***multivariable fuzzy model*** from numerical data using a self-organizing counterpropagation network. Both supervised and unsupervised learning algorithms are used for network training. Knowledge can be extracted from the data in the form of a set of rules. This rule base is then utilized by a fuzzy reasoning model. The rule base of the system, which is supposed to be relatively simple, is updated on-line in an adaptive way (in terms of connection weights) in response to the incoming data.

Cho and Wang (1996) developed an adaptive fuzzy system to extract the IF-THEN rules from sampled data through learning using a radial basis functions network. Different types of consequent, such as constants, first-order linear functions, and fuzzy variables are modelled, thereby enabling the network to handle arbitrary fuzzy inference schemes. There is not an initial rule base, and neither does one need to specify in advance the number of rules required to be identified by the system. Fuzzy rules are generated (when needed) by employing basis function units.

Wang and Mendel (1992a) described a fuzzy system by series of basis functions, which are algebraic superpositions of membership functions. Each such basis function corresponds to one fuzzy logic rule. An orthogonal least squares training algorithm is utilized to determine the significant fuzzy logic rules (structure learning) and associated parameters (parameter learning) from input-output training pairs. Owing to the possibility of acquiring and interpreting the linguistic IF-THEN rules by human experts, the fuzzy basis function network provides a framework for combining both numerical and linguistic information in a uniform manner.

Zhang and Morris (1999) used a recurrent neuro-fuzzy network to build long-term prediction models for nonlinear processes. Process knowledge is initially used to partition the process operation into several local fuzzy operating regions and also to set up the initial fuzzification layer weights. Membership functions of fuzzy operating regions are refined through training, enabling the local models to learn. The global model output is obtained by ***centre-of-gravity defuzzification*** involving the local models.

6.2.1 Fuzzy Neurons

The perceptron or processing unit described in Chapter 3, which employs multiplication, addition, and the sigmoid activation function to produce the nonlinear output from the applied input, is generally known as a *simple neural network*. However, if their architectures are extended by adding other mathematical

operations, such as triangular-norm, a triangular-co-norm, *etc*., to combine the incoming signals to the neuron, the extended networks give rise to a hybrid neural network based on fuzzy arithmetic operations. The fuzzy neural network architecture is practically based on such a processing element known as *fuzzy neuron* (Fuller,1995).

6.2.1.1 AND Fuzzy Neuron

Consider a perceptron-like structure as shown in Figure 6.4 with n input neurons acting as fan out elements (*i.e*. having the same output values as their inputs) and with one output neuron. The outputs x_i of the input-layer neurons are multiplied by the connecting weights w_i and, thereafter, fed to the output-layer neuron. If, however, the input signals x_i and the weights w_i are combined by an ***S-norm***, *i.e*. the ***triangular-conorm***

$$p_i = S\left(w_i, x_i\right), \quad i = 1, 2, \cdots, n. \tag{6.1}$$

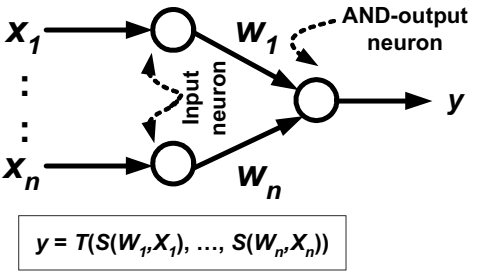

Figure 6.4. AND fuzzy neuron

and the input information p_i is further aggregated by a ***T-norm***, *i.e*. ***triangular norm***, to yield the final output of the neuron as

$$\begin{aligned} y &= AND\left(p_1, p_2, \ldots, p_n\right) = T\left(p_1, p_2, \ldots, p_n\right) \\ &= T\left(S\left(w_1, x_1\right), S\left(w_2, x_2\right), \ldots, S\left(w_n, x_n\right)\right). \end{aligned} \tag{6.2}$$

then the configuration in Figure 6.4 will represent the implementation of an ***AND fuzzy neuron*** under the condition that the *T-norm* represents a ***min operator*** and the *S-norm* represents a ***max operator***. Then the *min-max* composition

$$y = \min\left\{\max\left(w_1, x_1\right), \ldots, \max\left(w_n, x_n\right)\right\}. \tag{6.3}$$

can be realized by the AND fuzzy neuron.

6.2.1.2 OR Fuzzy Neuron

If a similar configuration to Figure 6.4 is used, but the signals x_i and the weights w_i are combined by a *triangular-norm* (*T-norm*)

$$p_i = T\left(w_i, x_i\right), \quad i = 1, 2, \cdots, n. \tag{6.4}$$

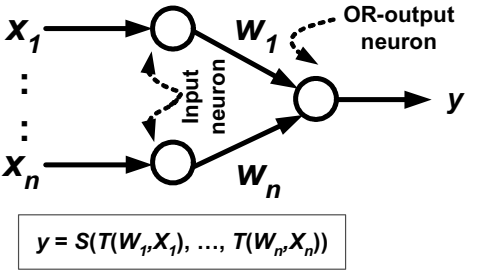

Figure 6.5. OR fuzzy neuron

and, thereafter, the input information p_i is further aggregated by a *triangular conorm* (*t-conorm* or *s-norm*) to yield the final output of the neuron as follows:

$$\begin{aligned} y &= OR\left(p_1, p_2, \ldots, p_n\right) = S\left(p_1, p_2, \ldots, p_n\right) \\ &= S\left(T\left(w_1, x_1\right), T\left(w_2, x_2\right), \ldots, T\left(w_n, x_n\right)\right). \end{aligned} \tag{6.5}$$

So, if the *t*-norm or the T = min operator and the *t-co norm* or the *s-norm* S = *max operator*, then the *max-min* composition can be realized by the *OR fuzzy neuron* as follows:

$$y = \max\left\{\min\left(w_1, x_1\right), \ldots, \min\left(w_n, x_n\right)\right\}. \tag{6.6}$$

Both fuzzy neurons realize logic operations on the membership values. The role of the connections is to differentiate between particular levels of impact that the individual inputs might have on the result of aggregation. We note that: (i) the higher the value of w_i the stronger is the impact of x_i on the output y of an *OR neuron*; (ii) the lower the value of w_i the stronger is the impact of x_i on the output y of an *AND neuron*.

The range of the output value y for the *AND neuron* is computed by letting all x_i equal to zero or one. By virtue of the monotonic property of the *triangular norms*, we obtain

$$y \in \left[T\left(w_1, \ldots, w_n\right), 1\right], \tag{6.7}$$

and for the OR neuron one derives the boundaries as

$$y \in \left[0, S\left(w_1, \ldots, w_n\right)\right]. \tag{6.8}$$

Similar to *AND* and *OR fuzzy neurons*, several other *fuzzy neurons*, such as *implication-OR*, *Kwan and Cai's fuzzy neuron*, *etc*. have been proposed (Fuller, 1995).

6.3 Neuro-fuzzy System Selection for Forecasting

The most common approach to numerical-data-driven neuro-fuzzy modelling is to use a Takagi-Sugeno-type fuzzy model along with differentiable operators and continuously differentiable membership functions (*e.g*. Gaussian function) for building the fuzzy inference mechanism, and the weighted average defuzzifier for defuzzification of output data. The corresponding output inference can then be represented in a multilayer feedforward network structure, such as the one depicted in Figure 6.6. In principle, the neuro-fuzzy network's architecture (Figure 6.6) is identical to the architecture of ANFIS, as shown in Figure 6.3.

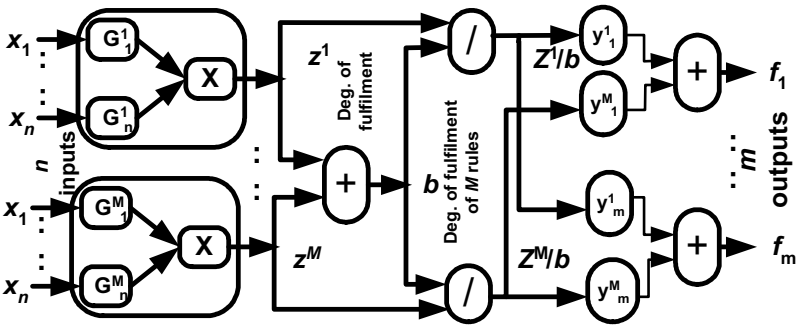

Figure 6.6. Fuzzy system as a multi-input multi-output feedforward neural network

The neuro-fuzzy model presented in Figure 6.6 is based on Gaussian membership functions. It uses Takagi-Sugeno-type fuzzy rules, product inference, and weighted average defuzzification. The nodes in the first layer calculate the degree of membership of the numerical input values in the antecedent fuzzy sets. The product nodes ($\times$) in the rectangular blocks (rounded corners) represent the antecedent conjunction operator and the output of this node is the corresponding degree of fulfilment $\left(z^l;\ l = 1,\ 2,\ 3, \cdots, M\right)$ or firing strength of the rule. The division nodes $(/)$, together with summation nodes (+), help implement the normalized

degree of fulfilment (z^l/b) of the corresponding rule, which, after multiplication with the corresponding Takagi-Sugeno rule consequent (y_m^l), is used as input to the summation block (+) at the final output layer. The output of this summation node is the final defuzzified output value, which, being crisp in nature, is directly compatible with the real world data. Once the fuzzy system of the above choice is represented as a feedforward network, the algorithm used for its training is less relevant.

A similar fuzzy model with singleton rule consequents, trained with standard backpropagation algorithm, was used by Wang and Mendel (1992b) for identification of various nonlinear plants.

Forecasting of time series is primarily based on numerical input-output data. To demonstrate this for neuro-fuzzy networks, a Takagi-Sugeno-type model, *i.e.* with linear rules consequent (and also a singleton model as a special case), is selected (Palit and Popovic, 1999; Palit and Babuška, 2001). Here, the number of membership functions to be implemented for fuzzy partitioning of input universes of discourse happens to be equal to the number of *a priori* selected fuzzy rules. To accelerate the convergence speed of the training algorithm and to avoid other inconveniences, the *Levenberg-Marquardt training algorithm* (described in the Section 6.4.2.3) or the *adaptive genetic algorithm* (AGA) can also be used.

In Chapter 4 it was shown that in forecasting of various nonlinear time series the fuzzy logic approach with automatically generated fuzzy rules (Wang and Mendel, 1992c; Palit and Popovic, 1999) works reasonably well. However, it was emphasized that the performance of fuzzy logic systems depends greatly on a set of well-consistent fuzzy rules and on the number of fuzzy membership functions implemented, along with their extent of overlapping. Therefore, determination of the optimum overlapping values of adjacent membership functions is very important in the sense that overlapping values too large or too small may deteriorate the forecasting accuracy. In the absence of firm guiding rules for optimum selection of overlapping, this selection mechanism was rather seen more as an art than as a science, mainly relying on a trial-and-error approach. Alternatively, very time-consuming heuristic approaches, such as the ***evolutionary computation*** or the ***genetic algorithms*** (Setnes and Roubos, 2000), can be used for this purpose.

Fuzzy logic systems encode numerical crisp values using linguistic labels, so it is difficult and time consuming to design and fine tune the membership functions related to such labels. However, neural networks' learning ability can automate this process. The combination of both fuzzy logic and neural network implementations can thus facilitate development of hybrid forecasters.

As an example we will consider the neural-networks-like architecture of the neuro-fuzzy system (Figure 6.6) and the training algorithm selected will fine tune the randomly generated system parameters. The great advantage of this scheme is that, apart from the user-selected number of fuzzy rules to be implemented, all other fuzzy parameters are automatically set by the training algorithm, so that the user does not need to bother about the optimal settings of fuzzy region overlappings and the like. Therefore, the approach to be described here is often

referred to as an ***adaptive neuro-fuzzy approach*** and the related fuzzy logic system as an ***adaptive fuzzy logic system*** (Wang, 1994).

6.4 Takagi-Sugeno-type Neuro-fuzzy Network

In the recent years much attention has been paid to deriving an effective data-driven neuro-fuzzy model because of its numerous advantages. For example, ANFIS-based (neuro-fuzzy) modelling was initially developed by Jang (1993) and Jang and Sun (1995), and later on widely applied in engineering. Similarly, singleton-rule-based and data-driven multi-input single output neuro-fuzzy modelling was initially developed by Wang and Mendel (1992b) and used for solving a variety of systems identification and control problems. A similar neuro-fuzzy network, with an improved training algorithm, was later developed and applied by Palit and Popovic (1999, 2000a, 2002b) and Palit and Babuška (2001) for time series forecasting. Because of its advantages compared with ANFIS, at least as far as model accuracy and the training time are concerned, this similar model, but with multi-input and multi-output structure, will be used in this chapter as a neuro-fuzzy forecaster. The advantages of this approach, where an explicitly Takagi-Sugeno-type multi-input multi-output fuzzy model is used, will be demonstrated on simulation examples of benchmark problems. Furthermore, the type of network selected can be regarded as a generalization or upgraded version of both a singleton-consequent-type multi-input single-output neuro-fuzzy network and the Takagi-Sugeno-type multiple input single output neuro-fuzzy network of Palit and Babuška (2001).

To avoid the fine tuning difficulties of initially chosen random membership functions, an efficient training algorithm for modelling of various nonlinear dynamics of multi-input multi-output systems is proposed that relies on a Takagi-Sugeno-type neuro-fuzzy network. The algorithm is further used for training the neuro-fuzzy network with the available data of a nonlinear electrical load time series. Thereafter, the trained network is used as a neuro-fuzzy model to predict the future value of electrical load data. In order to verify its prediction capability with other standard methods, some benchmark problems, such as Mackey-Glass chaotic time series and second-order nonlinear plant modelling, are considered.

Furthermore, the neuro-fuzzy approach described here attempts to exploit the merits of both neural-network and fuzzy-logic-based modelling techniques. For example, the fuzzy models are based on fuzzy IF-THEN rules and are, to a certain degree, transparent to interpretation and analysis, whereas the neural-networks-based black-box model has a unique learning ability.

In the following, the Takagi-Sugeno-type multiple-input multiple-output neuro-fuzzy system is constructed by multilayer feedforward network representation of the fuzzy logic system, as described in Section 6.4.1, and its training algorithm is described in Section 6.4.2. Thereafter, some comparisons between the radial basis function network and the proposed neuro-fuzzy network are made, followed by similar comparisons of the training algorithm for neural networks and neuro-fuzzy networks. Neuro-fuzzy modelling and time series forecasting are subsequently described and then, finally, some engineering examples are presented.

6.4.1 Neural Network Representation of Fuzzy Logic Systems

The fuzzy logic system considered here for constructing neuro-fuzzy structures is based on a Takagi-Sugeno-type fuzzy model with Gaussian membership functions. It uses product inference rules and a weighted-average defuzzifier defined as

$$f_j(x^p) = \sum_{l=1}^{M} y_j^l z^l \Big/ \sum_{l=1}^{M} z^l, \tag{6.9a}$$

where $j = 1, 2, 3, ..., m$; $l = 1, 2, 3, ..., M$;

$$y_j^l = \left(\theta_{0j}^l + \sum_{i=1}^{n} \theta_{ij}^l x_i\right), \text{ with } i = 1, 2, 3, \cdots, n; \tag{6.9b}$$

and $j = 1, 2, 3, ..., m$; $l = 1, 2, 3, ..., M$;

$$z^l = \prod_{i=1}^{n} \mu_{G_i^l}(x_i) \;;\; \mu_{G_i^l}(x_i) = \exp\left\{ -\left(x_i - c_i^l\right)^2 \Big/ \sigma_i^{l\,2} \right\} \tag{6.9c}$$

with $i = 1, 2, 3, ..., n$.

Here, we assume that $c_i^l \in U_i$, $\sigma_i^l > 0$, and $y_j^l \in V_j$, where U_i, and V_j are the input and output universes of discourse respectively. The corresponding lth rule from the above fuzzy logic system can be written as

R_l: *If* x_1 *is* G_1^l *and* x_2 *is* G_2^l *and ... and* x_n *is* G_n^l

Then $y_j^l = \theta_{0j}^l + \theta_{1j}^l x_1 + \theta_{2j}^l x_2 + \cdots + \theta_{nj}^l x_n$ (6.10)

where, x_i with $i = 1, 2, \ldots, n$; are the n system inputs, whereas f_j, with $j = 1, 2, \ldots, m$; are its m system outputs, and G_i^l, with $i = 1, 2, \ldots, n$; and $l = 1, 2, \ldots, M$; are the Gaussian membership functions of the form (6.9c) with the corresponding mean and variance parameters c_i^l and σ_i^l respectively and with y_j^l as the output consequent of the lth rule.

It is to be noted that the Gaussian membership functions (G_i^l) actually represent linguistic terms such as *low*, *medium*, *high*, *etc*. The rules (6.10), as specified above, are known as Takagi-Sugeno rules.

In the fuzzy logic system (6.9a) – (6.9c) the Gaussian membership function is deliberately chosen because the same membership function is continuously differentiable at all points. This is an essential requirement to apply the gradient-method-based training algorithm. Furthermore, it is also important to note that the fuzzy logic system (6.9a) – (6.9c) is capable of uniformly approximating any nonlinear function to any degree of accuracy over a universe of discourse $U \subset \mathbb{R}^n$ (Wang, 1994).

By carefully observing the functional forms (6.9a) – (6.9c), it can be seen that the above fuzzy logic system can be represented as a three-layer multi-input, multi-output feedforward network as shown in Figure 6.6. Because of the neuro implementation of the Takagi-Sugeno type fuzzy logic system, Figure 6.6 actually represents a Takagi-Sugeno-type of multi-input, multi-output neuro-fuzzy network, where, instead of the connection weights and the biases as in backpropagation neural networks, we have the mean (c_i^l) and the variance (σ_i^l) parameters of Gaussian membership functions, along with (θ_{0j}^l, θ_{ij}^l), *i.e.* y_j^l from the rules consequent, as the equivalent adjustable parameters of the network.

If the adjustable parameters of the neuro-fuzzy network are suitably selected, then the above fuzzy logic system can correctly approximate any nonlinear system based on given input–output data pairs.

6.4.2 Training Algorithm for Neuro-fuzzy Network

The fuzzy logic system, once represented as the equivalent multi-input, multi-output feedforward network (Figure 6.6), can generally be trained using any suitable training algorithm, such as the standard backpropagation algorithm (Palit *et al.*, 2002) that is generally used for neural networks training. However, because of its relatively slow speed of convergence, this algorithm needs to be further improved. Alternatively, a more efficient second-order training algorithm, such as the Levenberg-Marquardt algorithm described in the Section 6.4.2.3, can also be used.

6.4.2.1 Backpropagation Training of Takagi-Sugeno-type Neuro-fuzzy Network

Let a set of *N* input-output data pairs $\left(x^p, d_j{}^p\right)$, with p = 1, 2, 3, ..., *N*; and $x^p \equiv \left(x_1{}^p, x_2^p, \cdots, x_n^p\right) \in U \subset \mathbb{R}^n$, and $d_j{}^p \in V_j \subset \mathbb{R}^m$ is given. The objective is to determine a fuzzy logic system $f_j\left(x^p\right)$ in the form of (6.9a) – (6.9c), such that the performance function *S*, defined as

$$S_j = 0.5 \cdot \sum_{p=1}^{N} \left(e_j^p\right)^2 = 0.5 \cdot E_j^T E_j \tag{6.11a}$$

and

$$S = \sum_{j=1}^{m} S_j = \left(S_1 + S_2 + \cdots + S_m\right), \tag{6.11b}$$

is minimized, where E_j is the column vector of errors $e_j^p = \left\{f_j(x^p) - d_j{}^p\right\}$, and p = *1, 2, ..., N*; for the *j*th output from the fuzzy logic system. In addition, we also assume that the number of fuzzy rules and also the number of membership functions (to be implemented) *M* are given. In this way the problem is reduced to the adjustment of y_j^l, *i.e.* the parameters (θ_{0j}^l, θ_{ij}^l) from the rules consequent and

the mean (c_i^l) and variance (σ_i^l) parameters of the Gaussian membership functions, so that the performance function (6.11a) is minimized. For convenience, we replace $f_j\left(x^p\right)$, $d_j{}^p$, and e_j^p in the above definition of error by f_j, d_j, and e_j respectively, so that the individual error becomes $e_j = \left(f_j - d_j\right)$.

We recall that the *steepest descent rule* used for training of neuro-fuzzy networks is based on the recursive expressions

$$\theta_{0j}^l(k+1) = \theta_{0j}^l(k) - \eta \cdot \left(\partial S / \partial \theta_{0j}^l\right) \tag{6.12a}$$

$$\theta_{ij}^l(k+1) = \theta_{ij}^l(k) - \eta \cdot \left(\partial S / \partial \theta_{ij}^l\right) \tag{6.12b}$$

$$c_i^l(k+1) = c_i^l(k) - \eta \cdot \left(\partial S / \partial c_i^l\right) \tag{6.12c}$$

$$\sigma_i^l(k+1) = \sigma_i^l(k) - \eta \cdot \left(\partial S / \partial \sigma_i^l\right) \tag{6.12d}$$

where S is the performance function (6.11b) at the *kth* iteration step and $\theta_{0j}^l(k)$, $\theta_{ij}^l(k)$, $c_i^l(k)$, and $\sigma_i^l(k)$ are the free parameters of the network at the same iteration step, the starting values of which are, in general, randomly selected.

In addition, η is the constant step size or learning rate (usually $\eta << 1$), i = 1, 2, ..., n (with n as the number of inputs to the neuro-fuzzy network); j = 1, 2, ..., m (with m as the number of outputs from the neuro-fuzzy network); and l = 1, 2, 3, ..., M (with M as the number of Gaussian membership functions selected, as well as the number of fuzzy rules to be implemented).

From Figure 6.6, it is evident that the network output f_j and hence the performance function S_j and, therefore, finally S depends on θ_{0j}^l and θ_{ij}^l only through y_j^l. Similarly, the network output f_j and, thereby, the performance functions S_j and S depend on c_i^l and σ_i^l only through z^l, where, f_j, y_j^l, b, and z^l are represented by

$$f_j = \sum_{l=1}^{M} y_j{}^l \cdot h^l \tag{6.13a}$$

$$y_j{}^l = \theta_{0j}^l + \theta_{1j}^l x_1 + \theta_{2j}^l x_2 + \cdots + \theta_{nj}^l x_n \tag{6.13b}$$

$$h^l = \left(z^l / b\right), \text{and} \quad b = \sum_{l=1}^{M} z^l \tag{6.13c}$$

$$z^l = \prod_{i=1}^{n} \exp\left(-\left(\frac{x_i - c_i^l}{\sigma_i^l}\right)^2\right) \tag{6.13d}$$

Therefore, the corresponding chain rules

$$\left(\partial S/\partial \theta_{0j}^{l}\right)=\left(\partial S_j/\partial f_j\right)\cdot\left(\partial f_j/\partial y_j{}^{l}\right)\cdot\left(\partial y_j{}^{l}/\partial \theta_{0j}^{l}\right) \tag{6.14a}$$

$$\left(\partial S/\partial \theta_{ij}^{l}\right)=\left(\partial S_j/\partial f_j\right)\cdot\left(\partial f_j/\partial y_j{}^{l}\right)\cdot\left(\partial y_j{}^{l}/\partial \theta_{ij}^{l}\right) \tag{6.14b}$$

$$\begin{aligned}\left(\partial S/\partial c_i^{l}\right)&=\left(\partial\left(\sum_{j=1}^{m}S_j\right)\Big/\partial z^{l}\right)\cdot\left(\partial z^{l}/\partial c_i^{l}\right)\\&=\left(\sum_{j=1}^{m}\left(\partial S_j/\partial f_j\right)\cdot\left(\partial f_j/\partial z^{l}\right)\right)\cdot\left(\partial z^{l}/\partial c_i^{l}\right)\end{aligned} \tag{6.14c}$$

$$\begin{aligned}\left(\partial S/\partial \sigma_i^{l}\right)&=\left(\partial\left(\sum_{j=1}^{m}S_j\right)\Big/\partial z^{l}\right)\cdot\left(\partial z^{l}/\partial \sigma_i^{l}\right)\\&=\left(\sum_{j=1}^{m}\left(\partial S_j/\partial f_j\right)\cdot\left(\partial f_j/\partial z^{l}\right)\right)\cdot\left(\partial z^{l}/\partial \sigma_i^{l}\right)\end{aligned} \tag{6.14d}$$

can finally be written as

$$\left(\partial S/\partial \theta_{0j}^{l}\right)=\left(f_j-d_j\right)\cdot\left(z^{l}/b\right) \tag{6.15a}$$

$$\left(\partial S/\partial \theta_{ij}^{l}\right)=\left(f_j-d_j\right)\cdot\left(z^{l}/b\right)\cdot x_i \tag{6.15b}$$

$$\left(\partial S/\partial c_i^{l}\right)=A\cdot\left\{2\cdot\left(z^{l}/b\right)\cdot\left(x_i-c_i^{l}\right)\Big/\left(\sigma_i^{l}\right)^{2}\right\} \tag{6.15c}$$

$$\left(\partial S/\partial \sigma_i^{l}\right)=A\cdot\left\{2\cdot\left(z^{l}/b\right)\cdot\left(x_i-c_i^{l}\right)^{2}\Big/\left(\sigma_i^{l}\right)^{3}\right\} \tag{6.15d}$$

where,

$$\begin{aligned}A&=\left(\sum_{j=1}^{m}\left(y_j^{l}-f_j\right)\cdot\left(f_j-d_j\right)\right)\\&=\left(\sum_{j=1}^{m}\left(\left(\theta_{0j}^{l}+\sum_{i=1}^{n}\theta_{ij}^{l}\cdot x_i\right)-f_j\right)\cdot\left(f_j-d_j\right)\right)\end{aligned} \tag{6.15e}$$

Using the above results, the final update rules for the networks free parameters can be written as

$$\theta_{0j}^{l}(k+1)=\theta_{0j}^{l}(k)-\eta\cdot\left(f_j-d_j\right)\cdot\left(z^{l}/b\right) \tag{6.16a}$$

$$\theta_{ij}^{l}(k+1)=\theta_{ij}^{l}(k)-\eta\cdot\left(f_j-d_j\right)\cdot\left(z^{l}/b\right)\cdot x_i \tag{6.16b}$$

$$c_i^{l}(k+1)=c_i^{l}(k)-\eta.\left\{2\cdot A\cdot h^{l}\cdot\left(x_i-c_i^{l}\right)\Big/\left(\sigma_i^{l}\right)^{2}\right\} \tag{6.16c}$$

$$\sigma_i^{l}(k+1)=\sigma_i^{l}(k)-\eta\cdot\left\{2.A\cdot h^{l}\cdot\left(x_i-c_i^{l}\right)^{2}\Big/\left(\sigma_i^{l}\right)^{3}\right\}, \tag{6.16d}$$

where $h^l = \left(z^l / b\right)$, with $b = \sum_{l=1}^{M} z^l$ and h^l is the normalized degree of fulfilment (firing strength) of *l*th rule.

Equations (6.11a) – (6.16d) represent the *backpropagation training algorithm* (BPA) for Takagi-Sugeno-type multi-input multi-output neuro-fuzzy networks or the equivalent fuzzy logic system of form (6.9a) – (6.9c) with linear fuzzy rules consequent part as

$$y^l_j = \theta^l_{0j} + \theta^l_{1j} x_1 + \theta^l_{2j} x_2 + \theta^l_{3j} x_3 + \cdots + \theta^l_{nj} x_n .$$

In the above Takagi-Sugeno-type fuzzy rules (linear) consequent, if the coefficients $\theta^l_{ij} = 0$, for i = 1, 2, 3, ..., n; l =1, 2, 3, ..., M; and m = 1, then the equivalent neuro-fuzzy network is identical with the multi-input, single-output neuro-fuzzy network described by Wang and Mendel (1992b) and Palit and Popovic (1999 and 2000a). The resulting fuzzy logic system can be seen as a special case of both the Mamdani- and -Takagi-Sugeno-type systems, where the rule consequent is a singleton (constant number) fuzzy set. However, if $\theta^l_{ij} \neq 0$, for i = 1, 2, 3, ..., n; l= 1, 2, 3, ..., M; and for m = 1, then the resulting fuzzy logic system is identical with Takagi-Sugeno type multi-input and single-output neuro-fuzzy network, as described by Palit and Babuška (2001).

It is generally known that the backpropagation algorithm based on steepest descent rule, in order to avoid the possible oscillations in the final phase of the training, uses a relatively low learning rate $\eta << 1$. Therefore, the backpropagation training usually requires a large number of recursive steps or epochs. The acceleration of the training process with classical backpropagation, however, is achievable if the adaptive version of the learning rate or the momentum version of the steepest descent rule is used:

$$\theta^l_{0j}(k+1) = \theta^l_{0j}(k) - \eta \cdot (1 - mo) \cdot \left\{ \left(f_j - d_j\right)\left(z^l / b\right) \right\} + mo \cdot \Delta\theta^l_{0j}(k-1) \tag{6.17a}$$

$$\theta^l_{ij}(k+1) = \theta^l_{ij}(k) - \eta \cdot (1 - mo) \cdot \left\{ \left(f_j - d_j\right)\left(z^l / b\right) \right\} \cdot x_i + mo \cdot \Delta\theta^l_{ij}(k-1) \tag{6.17b}$$

$$c^l_i(k+1) = c^l_i(k) - \eta \cdot (1 - mo) \cdot \left\{ A \cdot 2 \cdot z^l \cdot \left(x_i - c^l_i\right) \Big/ \left(\sigma^l_i\right)^2 \right\} + mo \cdot \Delta c^l_i(k-1) \tag{6.17c}$$

$$\sigma^l_i(k+1) = \sigma^l_i(k) - \eta \cdot (1 - mo) \cdot \left\{ A \cdot 2 \cdot z^l \cdot \left(x_i - c^l_i\right)^2 \Big/ \left(\sigma^l_i\right)^3 \right\} + mo \cdot \Delta\sigma^l_i(k-1) \tag{6.17d}$$

where,

$$\Delta\mathbf{w}(k-1) = \mathbf{w}(k) - \mathbf{w}(k-1), \tag{6.17e}$$

and **w** represents the networks free parameter vector in general. The momentum constant is usually less than one. Therefore, we can write $mo < 1$.

6.4.2.2 Improved Backpropagation Training Algorithm

To improve the training performance of the proposed neuro-fuzzy network, we have modified the momentum version of the backpropagation algorithm by adding to it the modified error index term/modified performance index term (6.18a), as proposed by Xiaosong *et al.* (1995).

$$S_m(\mathbf{w}) = 0.5 \cdot \gamma \cdot \sum_{r=1}^{N} \left(e_r(\mathbf{w}) - e_{avg}\right)^2, \tag{6.18a}$$

where,

$$e_{avg} = \frac{1}{N} \cdot \sum_{r=1}^{N} e_r(\mathbf{w}). \tag{6.18b}$$

and e_{avg} is the average error. Thus, the new error index (new performance index) is finally defined as

$$S_{new}(\mathbf{w}) = S(\mathbf{w}) + S_m(\mathbf{w}), \tag{6.19}$$

where, $S(\mathbf{w})$ is the unmodified performance index as defined in (6.11b). From this, the corresponding gradient can be defined as

$$\nabla S(\mathbf{w}) = \sum_{r=1}^{N} (e_r(\mathbf{w})) \cdot (\partial e_r(\mathbf{w}) / \partial w) \tag{6.20a}$$

$$\nabla S_m(\mathbf{w}) = \sum_{r=1}^{N} \left(\gamma \cdot \left(e_r(\mathbf{w}) - e_{avg}\right)\right) \cdot (\partial e_r(\mathbf{w}) / \partial w) \tag{6.20b}$$

$$\nabla S_{new}(\mathbf{w}) = \sum_{r=1}^{N} \left(e_r(\mathbf{w}) + \gamma \cdot \left(e_r(\mathbf{w}) - e_{avg}\right)\right) \cdot (\partial e_r(\mathbf{w}) / \partial w), \tag{6.20c}$$

where the constant term (gama) $\gamma < 1$ has to be chosen appropriately.

With the modified error index extension as per Equation (6.20c) we need only to add a new vector term $\gamma \cdot \left(\mathbf{e}(\mathbf{w}) - e_{avg}\right)$ with the original error vector $\mathbf{e}(\mathbf{w})$. Theoretical justification of the improved training performance of the network by the use of a modified error index term has been described in Xiaosong *et al.* (1995).

6.4.2.3 Levenberg-Marquardt Training Algorithm

Training experiments with a neuro-fuzzy network using the momentum version of backpropagation algorithm, as well as its modified error index extension form, have shown that, with the first 200 training (four inputs- one output) data sets of a Mackey-Glass chaotic series, backpropagation algorithm usually requires several hundred epochs to bring the SSE value down to the desired error goal (see Palit and Popvic, 1999). This calls for an alternative, much faster training algorithm. Hence, to accelerate the convergence speed of neuro-fuzzy network training, the *Levenberg-Marquardt algorithm* (LMA) was proposed.

Although being an approximation to Newton's method, based on a Hessian matrix, the Levenberg-Marquardt algorithm can still implement the second-order training speed without direct computation of the Hessian matrix (Hagan and Menhaj, 1994). This is achieved in the following way.

Suppose that a function $V(\mathbf{w})$ is to be minimized with respect to the network's free-parameter vector $\mathbf{w}$ using Newton's method. The update of $\mathbf{w}$ to be used here is

$$\Delta\mathbf{w} = -\left[\nabla^2 V(\mathbf{w})\right]^{-1} \cdot \nabla V(\mathbf{w}) \tag{6.21a}$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w} \tag{6.21b}$$

where $\nabla^2 V(\mathbf{w})$ is the Hessian matrix and $\nabla V(\mathbf{w})$ is the gradient of $V(\mathbf{w})$. If the function $V(\mathbf{w})$ is taken to be the sum squared error function, *i.e.*

$$V(\mathbf{w}) = 0.5 \cdot \sum_{r=1}^{N} e_r^2(\mathbf{w}) \tag{6.22}$$

then the gradient $\nabla V(\mathbf{w})$ and the Hessian matrix $\nabla^2 V(\mathbf{w})$ are generally defined using the Jacobian matrix $J(\mathbf{w})$ as

$$\nabla V(\mathbf{w}) = J^T(\mathbf{w}) \cdot e(\mathbf{w}) \tag{6.23a}$$

$$\nabla^2 V(\mathbf{w}) = J^T(\mathbf{w}) \cdot J(\mathbf{w}) + \sum_{r=1}^{N} e_r(\mathbf{w}) \cdot \nabla^2 e_r(\mathbf{w}), \tag{6.23b}$$

where

$$J(\mathbf{w}) = \begin{bmatrix} \frac{\partial e_1(\mathbf{w})}{\partial w_1} & \frac{\partial e_1(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_1(\mathbf{w})}{\partial w_{N_p}} \\ \frac{\partial e_2(\mathbf{w})}{\partial w_1} & \frac{\partial e_2(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_2(\mathbf{w})}{\partial w_{N_p}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_N(\mathbf{w})}{\partial w_1} & \frac{\partial e_N(\mathbf{w})}{\partial w_2} & \cdots & \frac{\partial e_N(\mathbf{w})}{\partial w_{N_p}} \end{bmatrix} \tag{6.23c}$$

and $w = \left[w_1,\ w_2,\ \cdots, w_{N_p} \right]$ is the parameter vector of network. From (6.23c) it is seen that the dimension of the Jacobian matrix is ($N \times N_p$), N and N_p being the number of training samples and the number of adjustable network parameters respectively. For the Gauss-Newton method the second term in (6.23b) is assumed to be zero, so that the update according to (6.21a) becomes

$$\Delta\mathbf{w} = -\left[J^T(\mathbf{w}) \cdot J(\mathbf{w})\right]^{-1} \cdot J^T(\mathbf{w}) \cdot e(\mathbf{w}). \tag{6.24a}$$

The Levenberg-Marquardt modification of the Gauss-Newton method is

$$\Delta\mathbf{w} = -\left[J^T(\mathbf{w}) \cdot J(\mathbf{w}) + \mu \cdot I\right]^{-1} \cdot J^T(\mathbf{w}) \cdot e(\mathbf{w}) \tag{6.24b}$$

in which I is the ($N_p \times N_p$) identity matrix and the parameter μ is multiplied by some constant factor μ_{inc} whenever an iteration step increases the value of $V(\mathbf{w})$, and divided by μ_{dec} whenever a step reduces the value of $V(\mathbf{w})$. Hence, the update according to (6.21b) is

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \left[J^T(\mathbf{w}) \cdot J(\mathbf{w}) + \mu \cdot I \right]^{-1} \cdot J^T(\mathbf{w}) \cdot e(\mathbf{w}). \tag{6.24c}$$

Note that for large μ the algorithm becomes the steepest descent gradient algorithm with step size $(1/\mu)$, whereas for small μ, *i.e.* $\mu \approx 0$, it becomes the Gauss-Newton algorithm. Usually, $\mu_{inc} = \mu_{dec}$. However, in our program we have selected two different values for them. In order to get even faster convergence, a small momentum term mo = 0.098 was also added, so that the final update becomes

$$\begin{aligned} \mathbf{w}(k+1) = \mathbf{w}(k) &- \left[J^T(\mathbf{w}) \cdot J(\mathbf{w}) + \mu \cdot I \right]^{-1} \cdot J^T(\mathbf{w}) \cdot e(\mathbf{w}) \\ &+ mo \cdot \left(\mathbf{w}(k) - \mathbf{w}(k-1) \right) \end{aligned} \tag{6.24d}$$

It is to be noted that the use of a momentum term is quite usual with the classical backpropagation algorithm, whereas this may appear to be unusual with the Levenberg-Marquardt algorithm. However, the latter is justified, as the use of a momentum term in the backpropagation algorithm is primarily to overcome the possible trap at local minima and also to prevent small oscillations during the training of the network; similarly, the use of a small momentum term, as experimentally verified through simulation, also helps to increase network training convergence with the Levenberg-Marquardt algorithm. Furthermore, similar to the backpropagation algorithm, here also the Levenberg-Marquardt algorithm was extended by adding a modified error index term, as proposed by Xiaosong *et al*. (1995), to improve further the training convergence. Therefore, as per (6.20c), the corresponding new gradient can now be expressed or defined using a Jacobian matrix as

$$\nabla S_{new}(\mathbf{w}) = J^T(\mathbf{w}) \cdot \left[\mathbf{e}(\mathbf{w}) + \gamma \cdot \left(\mathbf{e}(\mathbf{w}) - e_{avg}\right)\right], \tag{6.25}$$

where **e(w)** represents the column vector of errors, and the constant factor $\gamma << 1$ (for the Levenberg-Marquardt algorithm) has to be chosen appropriately. Equation (6.25) suggests that even with consideration of the modified error index extension of the original performance function the Jacobian matrix remains unaltered and, with the above modification, we need to add only a new error vector term $\gamma \cdot \left(\mathbf{e}(\mathbf{w}) - e_{avg}\right)$ with the original error vector $\mathbf{e}(\mathbf{w})$ as we did with the back-propagation algorithm.

6.4.2.3.1 Computation of Jacobian Matrix

We now describe a simplified technique to compute, layer by layer, the Jacobian matrix and the related parameters from the backpropagation results. Layer-wise or parameter-wise computation of the Jacobian matrix is permissible because, as stated in Equations (6.26a) and (6.26b), the final contents of the Hessian matrix remain unaltered even if the whole Jacobian is divided into smaller parts. Furthermore, this division of the Jacobian matrix helps to avoid computer memory shortage problem, which is likely to occur for large neural networks.

From

$$\nabla^2 V(w) \approx \left[J^T(w)\right] \cdot \left[J(w)\right] = \left[J_1^T(w), J_2^T(w)\right] \cdot \begin{bmatrix} J_1(w) \\ J_2(w) \end{bmatrix} \tag{6.26a}$$

it follows that

$$\nabla^2 V(w) \approx \left[J_1^T(w) \cdot J_1(w) + J_2^T(w) \cdot J_2(w)\right]. \tag{6.26b}$$

Computation of the Jacobian matrix is in fact the most crucial step in implementing the Levenberg-Marquardt algorithm for neuro-fuzzy networks. For this purpose,

the results obtained in the Section 6.4.2.1 will be used, where the derivatives of the sum square error S with respect to the network's adjustable parameters (free-parameters) θ_{0j}^{l}, θ_{ij}^{l}, c_i^l, and σ_{i}^{1} for the fuzzy logic system (6.9a) – (6.9c) were already computed and listed in (6.15a) – (6.15e).

Now, considering the singleton consequent part (constant term) of the rules and taking into account Equation (6.15a), we can rewrite the gradient $V\left(\theta_{0j}^{l}\right) \equiv S$ as

$$\nabla V\left(\theta_{0j}^{l}\right) \equiv \left(\partial S / \partial \theta_{0j}^{l}\right) = \left\{z^l / b\right\} \cdot \left(f_j - d_j\right), \tag{6.27}$$

where f_j is the actual output vector from the jth output node of the Takagi-Sugeno-type multiple input multiple output neuro-fuzzy network and d_j is the corresponding desired output vector at the jth output node for a given set of input-output training data. Taking into account Equation (6.27) and comparing it with (6.23a), where the gradient is expressed using the transpose of the Jacobian matrix multiplied by the network's error vector, *i.e.*

$$\nabla V(\mathbf{w}) = J^T(\mathbf{w}) \cdot e(\mathbf{w}), \tag{6.28}$$

where $\mathbf{w}$ is the free parameter of the network, the transpose of the Jacobian matrix $J^T\left(\theta_{0j}^{l}\right)$ and the Jacobian matrix $J\left(\theta_{0j}^{l}\right)$ for the free parameter θ_{0j}^{l} of the neuro-fuzzy network can be defined by

$$J^T\left(\theta_{0j}^{l}\right) = \left(z^l / b\right) \tag{6.29a}$$

$$J\left(\theta_{0j}^{l}\right) \equiv \left[J^T\left(\theta_{0j}^{l}\right)\right]^T = \left[z^l / b\right]^T. \tag{6.29b}$$

This is because the prediction error at the jth output node of the Takagi-Sugeno-type neuro-fuzzy network is

$$e_j \equiv \left(f_j - d_j\right). \tag{6.30}$$

However, if we consider the normalized prediction error of the network at the jth output node, instead of the original prediction error at the jth output node, then by applying a similar technique, the transposition of the Jacobian matrix $J^T\left(\theta_{0j}^{l}\right)$ and the Jacobian matrix $J\left(\theta_{0j}^{l}\right)$ itself for the free parameter θ_{0j}^{l} will be

$$J^T\left(\theta_{0j}^{l}\right) = \left(z^l\right) \tag{6.31a}$$

$$J\left(\theta_{0j}^{l}\right) \equiv \left[J^T\left(\theta_{0j}^{l}\right)\right]^T = \left[z^l\right]^T, \tag{6.31b}$$

this is because the normalized prediction error at the *j*th output node of the multi-input multi-output neuro-fuzzy network is:

$$e_j(\text{normalized}) \equiv \left(f_j - d_j\right)\!/b\,. \tag{6.32}$$

In the above equation, z^l is a matrix of size $(M \times N)$ that contains the degree of fulfilment (firing strength) of each fuzzy rule computed for a given set of training samples, where *M* is the number of fuzzy rules (and also the number of Gaussian membership functions implemented for fuzzy partition of input universes of discourse) and *N* is the number of training samples (input-output data samples).

Adopting a similar technique and taking into account Equation (6.28), the original prediction error (6.30) and Equation (6.15b), which computes the derivative of *S* with respect to θ_{ij}^l, we can get the transposition of the Jacobian matrix and its further transposition, *i.e.* the Jacobian matrix itself, for the network's free-parameter θ_{ij}^l using

$$J^T\left(\theta_{ij}^l\right) = \left(z^l/b\right)\cdot x_i \tag{6.33a}$$

$$J\left(\theta_{ij}^l\right) \equiv \left[J^T\left(\theta_{ij}^l\right)\right]^T = \left[\left(z^l/b\right)\cdot x_i\right]^T. \tag{6.33b}$$

Also, instead of the original prediction error, if here we consider the normalized prediction error of Equation (6.32) and, as usual, Equations (6.28) and (6.15b), then we can get the transposed Jacobian matrix and the Jacobian matrix itself for the same parameter θ_{ij}^l as

$$J^T\left(\theta_{ij}^l\right) = \left(z^l \cdot x_i\right) \tag{6.34a}$$

$$J\left(\theta_{ij}^l\right) \equiv \left[J^T\left(\theta_{ij}^l\right)\right]^T = \left[z^l \cdot x_i\right]^T \tag{6.34b}$$

Finally, to compute the Jacobian matrices and their transpositions for the remaining free parameters of the network, *i.e.* for parameters c_i^l, and σ_i^l, we also use a similar technique, whereby Equation (6.15e), which computes the term *A*, has to be reorganized.

Let us denote

$$D_j \equiv \left(y^l{}_j - f_j\right). \tag{6.35}$$

Using Equations (6.30) and (6.35) we can rewrite (6.15e) as

$$A \equiv \sum_{j=1}^{m} \left(D_j \cdot e_j \right) = \left(D_1 \cdot e_1 + D_2 \cdot e_2 + \cdots + D_m \cdot e_m \right) \tag{6.36}$$

Our objective is to find suitable terms D_{eqv} and e_{eqv} such that their product is equal to

$$A \equiv D_{\text{eqv}} \cdot e_{\text{eqv}} = \left(D_1 \cdot e_1 + D_2 \cdot e_2 + \cdots + D_m \cdot e_m \right) \tag{6.37}$$

where the term e_{eqv} is such that it contributes the same amount of sum squared error value S of equation (6.11b) as that can be obtained jointly by all the $e_j \equiv \left(f_j - d_j \right)$ from the multiple-input multiple-output network. Therefore,

$$e_{\text{eqv}}^{p} = \sqrt{\left(e_1^{p^2} + e_2^{p^2} + \cdots + e_m^{p^2} \right)}, \tag{6.38}$$

where, $p = 1, 2, 3, \ldots, N$; corresponding to N training samples. This results in

$$D_{\text{eqv}} = A \cdot \left(e_{\text{eqv}} \right)^{-1} \tag{6.39a}$$

This can be written in matrix form using the pseudo inverse as

$$\mathbf{D}_{\text{eqv}} = \mathbf{A} \cdot \left(\mathbf{E}_{\text{eqv}} \right)^{T} \cdot \left(\mathbf{E}_{\text{eqv}} \cdot \mathbf{E}_{\text{eqv}}^{T} \right)^{-1} \tag{6.39b}$$

where $\mathbf{E}_{\text{eqv}}$ is the equivalent error vector of size $\left(N \times 1 \right)$ containing $\left(e_{\text{eqv}}^{p} \right)$ as its elements for all (N) training samples. Similarly, $\mathbf{D}_{\text{eqv}}$ and $\mathbf{A}$ are matrices of size $\left(M \times N \right)$ and $\left(M \times 1 \right)$ respectively. Once the matrix $\mathbf{D}_{\text{eqv}}$ and the equivalent error vector $\mathbf{E}_{\text{eqv}}$ are known, we can replace matrix $\mathbf{A}$ with their product. Therefore,

$$\mathbf{A} = \mathbf{D}_{\text{eqv}} \cdot \mathbf{E}_{\text{eqv}} \tag{6.40a}$$

or, equivalently as,

$$A = D_{\text{eqv}} \cdot e_{\text{eqv}} \tag{6.40b}$$

can be calculated. In the case of a multiple-input single-output neuro-fuzzy network, *i.e.* for $m = 1$ and $A = D_1 \cdot e_1$, $D_{\text{eqv}} = D_1$ and $e_{\text{eqv}} = e_1$ hold. This means that, in this case, Equations (6.37) – (6.40b) need not be computed.

However, for the multiple-input multiple-output case, where $m \geq 2$, using (6.37) we can write Equations (6.15c) and (6.15d) as

$$\left(\partial S/\partial c_i^l\right)=\left(D_{\text{eqv}}\cdot e_{\text{eqv}}\right)\cdot\left\{2\cdot\left(z^l/b\right)\cdot\left(x_i-c_i^l\right)\Big/\left(\sigma_i^l\right)^2\right\} \tag{6.41a}$$

$$\left(\partial S/\partial \sigma_i^l\right)=\left(D_{\text{eqv}}\cdot e_{\text{eqv}}\right)\cdot\left\{2\cdot\left(z^l/b\right)\cdot\left(x_i-c_i^l\right)^2\Big/\left(\sigma_i^l\right)^3\right\} \tag{6.41b}$$

Now, following the previous technique and realizing that $\left(e_{\text{eqv}}/b\right)$ can be considered as the normalized equivalent error and, in addition, taking into account Equation (6.28) and comparing it respectively with (6.41a) and (6.41b), transposed Jacobian matrix and the Jacobians $J^T\left(c_i^l\right)$, $J\left(c_i^l\right)$ and $J^T\left(\sigma_i^l\right)$, $J\left(\sigma_i^l\right)$ for the network free parameters c_i^l and σ_i^l can be computed as:

$$J^T\left(c_i^l\right)=\left\{2\cdot D_{eqv}\cdot z^l\cdot\left(x_i-c_i^l\right)\Big/\left(\sigma_i^l\right)^2\right\} \tag{6.42a}$$

$$J\left(c_i^l\right)=\left[J^T\left(c_i^l\right)\right]^T=\left[2\cdot D_{eqv}\cdot z^l\cdot\left(x_i-c_i^l\right)\Big/\left(\sigma_i^l\right)^2\right]^T \tag{6.42b}$$

$$J^T\left(\sigma_i^l\right)=\left\{2\cdot D_{eqv}\cdot z^l\cdot\left(x_i-c_i^l\right)^2\Big/\left(\sigma_i^l\right)^3\right\} \tag{6.42c}$$

$$J\left(\sigma_i^l\right)=\left[J^T\left(\sigma_i^l\right)\right]^T=\left[2\cdot D_{eqv}\cdot z^l\cdot\left(x_i-c_i^l\right)2\Big/\left(\sigma_i^l\right)^3\right]^T \tag{6.42d}$$

The above equations describe the Jacobian matrices and their transpositions for the Takagi-Sugeno-type fuzzy logic systems with the adjustable free parameters c_i^l and σ_i^l when normalized (equivalent) error is considered.

If, however, instead of normalized (equivalent) error only the equivalent error is considered, then the Jacobian matrices and their transpositions will be the same, except that in the right-hand sides of Equations (6.42a) – (6.42c) the term z^l has to be replaced by normalized degree of fulfilment of the lth rule $h^l=\left(z^l/b\right)$, where $b=\sum_{l=1}^{M}z^l$ represents the sum of degree of fulfilment of all rules.

It is to be noted that, while computing the Jacobian matrices, care has to be taken so that the dimensions of the Jacobians match correctly with $\left(N\times N_p\right)$, where N is the number of training data sets and N_p the number of adjustable parameters in the network's layer considered. In all our simulation experiments with neuro-fuzzy networks the *normalized prediction error* has been considered for the computation of Jacobian matrices for the network's free parameters θ_{0j}^l and θ_{ij}^l, so that Equations (6.31a), (6.31b) and Equations (6.34a), (6.34b) delivered the corresponding transposed Jacobian matrices and their Jacobians respectively. In contrast, *normalized equivalent error* has been considered for the computation of transposed Jacobian matrices and their Jacobians respectively for the mean and

variance parameters c_i^l and σ_i^l of the Gaussian membership functions; therefore, Equations (6.42a) – (6.42c) delivered the corresponding transpositions of the Jacobian matrices and the Jacobian matrices themselves for the Takagi-Sugeno-type multi-input, multi-output neuro-fuzzy network's free parameter and gave the Levenberg-Marquardt algorithm better convergence in most experiments.

6.4.2.4 Adaptive Learning Rate and Oscillation Control

The proposed backpropagation training algorithm and the Levenberg-Marquardt training algorithm, both with the modified error index extension as performance function and with the added small momentum term, have proven to be very efficient, and faster in training the Takagi-Sugeno-type neuro-fuzzy networks than the standard back-propagation algorithm. But still, the performance function of the network (if left without any proper care) is not always guaranteed to reduce, in every epoch, towards the desired error goal. As a consequence, the training can proceed in the opposite direction, giving rise to a continuous increase of performance function or to its oscillation. This prolongs the training time or makes the training impossible. To avoid this, three sets of adjustable parameters are recommended to be stored for the backpropagation algorithm and two sets for the Levenberg-Marquardt algorithm. The stored sets are then used in the following way.

In the case of the backpropagation algorithm, if two consecutive new sets of adjustable parameters reduce the network performance function, then in the following epochs the same sets are used and the learning rate in the next step is increased slightly by a factor of 1.1. In the opposite case, *i.e.* if the performance function with the new sets of parameters tends to increase beyond a given limit - say *WF* (*wildness factor* of oscillation) times the current value of the performance function – then the new sets are discarded and training proceeds with the old sets of adjustable parameters. Thereafter, a new direction of training is sought with the old sets of parameters and with lower values of the learning rate parameter, *e.g.* 0.8 or 0.9 times the old learning rate.

In the case of the Levenberg-Marquardt algorithm, if the following epoch reduces the value of the performance function, then the training proceeds with a new set of parameters and the μ value is reduced by a preassigned factor $(1/\mu_{dec})$. In the opposite case, *i.e.* if the next epoch tends to increase this performance value beyond the given limits (*WF* times of current value of performance function) or remains the same, then the μ value is increased by another preassigned factor (μ_{inc}) but the new set of adjustable parameters is discarded and training proceeds with the old set of parameters. In this way, in every epoch the value of the performance function is either decreased steadily or at least maintained within the given limit values.

6.5 Comparison of Radial Basis Function Network and Neuro-fuzzy Network

There are considerable similarities, as well as dissimilarities, between the RBF-type neural network and neuro-fuzzy network. In this section we present a few comparisons between them.

A radial basis function network can be considered as a three-layer network consisting of an input layer, a hidden layer and an output layer (see Chapter 3 for details). The hidden layer performs the nonlinear transformation, so that the input space is mapped into a new space. The output layer then combines the outputs of the hidden layer linearly. The structure of an RBF network with an input vector $\mathbf{x} \in \mathbb{R}^n$ and output $y \in \mathbb{R}$ is shown in Chapter 3. The output from such a network can be written as

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i R_i(\mathbf{x}),$$

where w_i are the weights and $R_i(\mathbf{x})$ is the nonlinear activation function of the hidden-layer neurons.

The fuzzy logic system considered in Equations (6.9a) – (6.9c) can be rewritten as

$$f_j(x^p) = \sum_{l=1}^{M} y_j^l \cdot h^l \quad \text{where} \quad h^l = z^l \Big/ \sum_{l=1}^{M} z^l,$$
$$\text{and} \quad j = 1, 2, 3, \cdots, m; \quad l = 1, 2, 3, \cdots, M.$$

noting that, when using the definition of the radial basis function the normalized degree of fulfilment of the lth rule, *i.e.* $h^l \equiv h^l(x_i)$, is similar to an RBF. Therefore, the fuzzy logic system can also be represented as an RBF neural network model. However, the following points have to be carefully noted:

- Functions in the form of (6.9a) are just one kind of fuzzy logic system with a particular choice of fuzzy inference engine with *product inference rules*, a fuzzifier, and a weighted-average defuzzifier. If another choice is made, such as the *mean-of-maxima (MOM) defuzzifier*, then the fuzzy logic system will be quite different from the RBF network. Therefore, an RBF network in fact is a special case of the fuzzy logic system.

- The membership functions of the fuzzy logic system can take various geometric forms (such as Gaussian, triangular, trapezoidal, bell-shaped, *etc.*). They can also be non-homogeneous (*i.e.* the membership functions that divide the input or output universe of discourse may not all be of the same functional form), whereas the RBF network takes a lesser number of functional forms, like a Gaussian function, and are usually homogeneous. This is due to the different justifications of the neuro-fuzzy network and

the RBF networks. The fuzzy logic systems are justified from the human reasoning point of view and, therefore, the membership functions can have any suitable form within the range [0, 1], appropriate to representing the knowledge of a human expert through IF-THEN rules. On the other hand, RBF networks are based on biological motivations. Therefore, it is difficult to justify the use of many different kinds of non-homogeneous basis functions in a single RBF network.

One of the fundamental differences between a neuro-fuzzy network and an RBF network is that the former takes the linguistic information explicitly into consideration and makes use of it in a systematic manner, whereas the latter does not. Furthermore, while using the neuro-fuzzy network, besides the generated model accuracy we are also concerned about the transparency of the model, whereas for the RBF network, and also for other types of neural network, we are only concerned about the model accuracy (black-box modelling).

6.6 Comparison of Neural Network and Neuro-fuzzy Network Training

We would now like to compare the back-propagation training algorithms for the multi-layer perceptron networks and neuro-fuzzy networks described in this chapter. The training algorithms are similar in the following sense:

- Their basic operation, *i.e.* forward computation and backward training, is the same, and in order to minimize the sum squared error between the actual output and the desired output of the network, both of them use either the same gradient method or the second-derivative-based recursive algorithm, *i.e.* the approximate Hessian matrix.

- Both of them are universal approximators and, therefore, well qualified to solve any nonlinear mapping to any degree of accuracy within the universe of discourse.

However, they differ distinctly in the following:

- The parameters (weights and biases) of the neural networks have no clear physical meaning or interpretation (black-box modelling), which makes the selection of their initial values difficult; thus, they are chosen rather randomly. On the other hand, the parameters of the neuro-fuzzy networks have clear physical meaning (membership functions), so that if the sufficient knowledge about the system to be modelled by the neuro-fuzzy networks is available, then a good initial parameter setting procedure can be developed.

- Besides numerical information, linguistic information can also be incorporated into neuro-fuzzy systems.

6.7 Modelling and Identification of Nonlinear Dynamics

We would now like to illustrate the efficiency of the neuro-fuzzy approach proposed in Section 6.4.1 on some forecasting examples.

6.7.1 Short-term Forecasting of Electrical Load

This application concerns the forecasting the electrical load demand, based on a time series that predicts the values at time ($t + L$) using the available observation data up to the time point t. For modelling purposes the time series data $\mathbf{X} = \{X_1, X_2, X_3, \ldots, X_q\}$ have been rearranged in input-output form **XIO**. The neuro-fuzzy predictor to be developed for time series modelling and forecasting is supposed to operate with four inputs (*i.e.* $n = 4$) and with three outputs (*i.e.* $m = 3$). Taking both the sampling interval and the lead time of forecast to be one time unit, then for each $t \geq 4$ the input data have to be represented as a four-dimensional vector and the output data as a three-dimensional vector

$$\text{XI} = [X(t\text{-}3), X(t\text{-}2), X(t\text{-}1), X(t)],$$
$$\text{XO} = [X(t\text{+}1), X(t\text{+}2), X(t\text{+}3)]$$

Furthermore, in order to have sequential output in each row, the values of t should run as 4, 7, 10, 13, …, (q-3). The corresponding **XIO** matrix will then look like (6.43), in which the first four columns represent the four inputs of the network and the last three columns represent its output.

$$\text{XIO} = \begin{bmatrix} X_1, & X_2, & X_3, & X_4 \Rightarrow X_5, & X_6, X_7 \\ X_4, & X_5, & X_6, & X_7 \Rightarrow X_8, & X_9, X_{10} \\ \vdots & \vdots & \vdots & \vdots & \vdots \quad \vdots \quad \vdots \\ X_{q-6}, & X_{q-5}, & X_{q-4}, & X_{q-3} \Rightarrow X_{q-2}, & X_{q-1}, X_q \end{bmatrix} \tag{6.43}$$

In the selected forecasting example, 1163 input-output data were generated, from which only the first 500 input-output data sets, *i.e.* the first 500 rows from the XIO matrix, were used for the multi-input multi-output neuro-fuzzy network training. The remaining 663 rows of the XIO matrix were used for verification of the forecasting results. The training and forecasting performances achieved with the neuro-fuzzy network are illustrated in Figures 6.7(a) – (d) and in Tables 6.1(a) and 6.1(b) respectively.

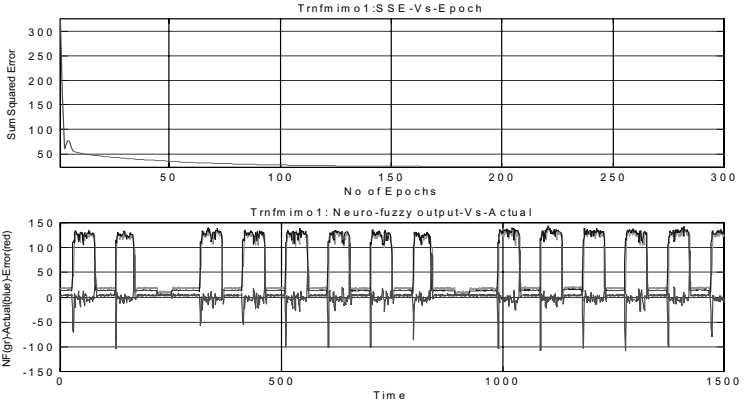

Figure 6.7(a). Training performance of Takagi-Sugeno-type multi-input multi-output neuro-fuzzy network with $n = 4$ inputs, $m = 3$ outputs, $M = 15$ fuzzy rules and 15 GMFs for short-term forecasting of electrical load time series when trained with proposed backpropagation algorithm. Backpropagation algorithm training parameters: $\eta = 0.0005$, $\gamma = 0.5$, $mo = 0.5$, maximum epoch = 300, training (pre-scaled) data = 1 to 500 rows of XIO matrix, initial SSE = 324.6016 (with random starting parameter), final SSE = 23.8580, data scaling factor = 0.01.

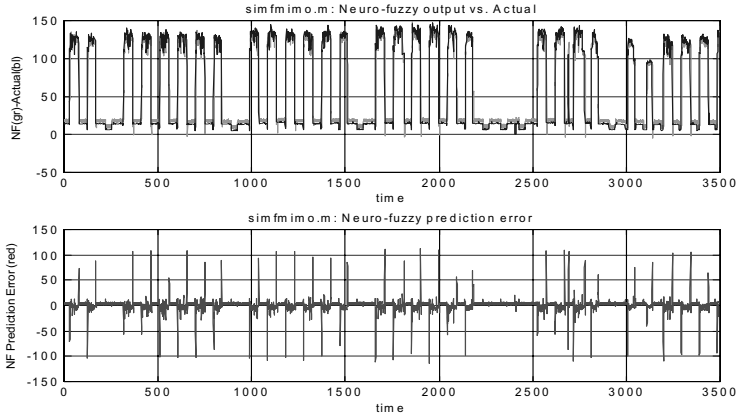

Figure 6.7(b). Forecasting performance of Takagi-Sugeno-type multi-input multi-output neuro-fuzzy network with $n = 4$ inputs, $m = 3$ outputs, $M = 15$ fuzzy rules and 15 GMFs for short-term forecasting of electrical load when trained with proposed backpropagation algorithm. Data 1 to 1500 correspond to training data and data 1501 to 3489 (*i.e.* row 501 to 1163 from XIO matrix) represent the forecasting performance.

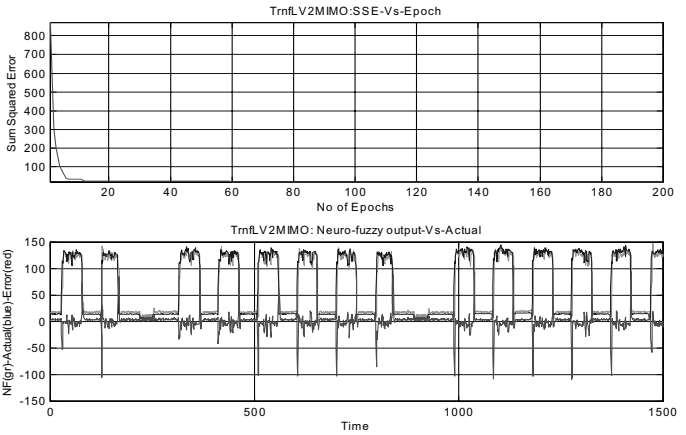

Figure 6.7(c). Training performance of the Takagi-Sugeno-type multi-input multi-output neuro-fuzzy network with $n = 4$ inputs, $m = 3$ outputs, $M = 15$ fuzzy rules and 15 GMFs for short-term forecasting of electrical load time series with the proposed Levenberg-Marquardt algorithm. Training parameters of Levenberg-Marquardt algorithm: $\mu = 0.001$, $\gamma = 0.1$, $mo = 0.1$, maximum epoch = 200, training (pre-scaled) data = 1 to 500 rows of XIO matrix, initial SSE = 868.9336 (with random starting parameter of neuro-fuzzy network), final SSE = 22.5777, data scaling factor = 0.01.

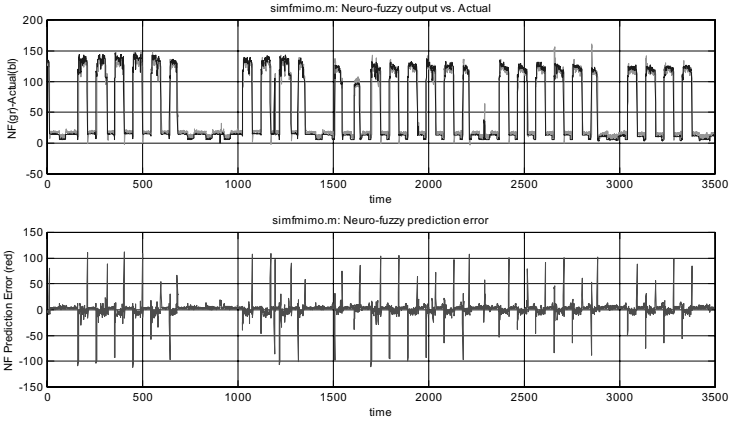

Figure 6.7(d). Forecasting performance of the Takagi-Sugeno-type multi-input multi-output neuro-fuzzy network with $n = 4$ inputs, $m = 3$ outputs, $M = 15$ fuzzy rules and 15 GMFs for short-term forecasting of electrical load after the training with the proposed Levenberg-Marquardt algorithm. Note that in both Figures 6.7(b) and 6.7(d) data from 1 to 1500 correspond to training data and data from 1501 to 3489 represent the forecasting performance with validation data set. It is important to note that data within the time points 2200 to 2510 are different from the training data. Still the Takagi-Sugeno-type multi-input

multi-output neuro-fuzzy network can predict this data region with reasonably high accuracy.

Table 6.1(a). Training and forecasting performance of Takagi-Sugeno-type multi-input multi-output neuro-fuzzy network with proposed backpropagation algorithm for electrical load time series.

Sl. No.	Final SSE with pre-scaled data (scale factor = 0.001)	Final SSE, MSE, RMSE with original (nonscaled) data
1.	SSE = 23.8580 SSE1 = 3.0077, SSE2 = 7.2863, SSE3 = 13.5640 (with training data 1 to 1500) (After training with backpropagation algorithm)	SSE = 2.3858e+005 (with training data 1 to 1500) MSE1 = 30.0772, MSE2 = 72.8630, MSE3 = 135.6401; RMSE1 = 5.4843, RMSE2 = 8.5360, RMSE3 = 11.6465
2.	SSE = 53.5633; SSE1 = 6.8169, SSE2 = 16.6395, SSE3 = 30.1069, (with training and validation data points 1 to 3489)	SSE = 5.3563e+005 (with training and validation data points 1 to 3489)

Figure 6.7(a) and Table 6.1(a) demonstrate that the proposed backpropagation algorithm brings the sum squared error as the performance function smoothly from its initial value of 324.6016 down to 23.8580 in 300 epochs, whereas Figure 6.7(c) and Table 6.1(b) demonstrate the training performance with the proposed Levenberg-Marquardt algorithm. In the latter case the performance function is brought down to 22.5777 from its initial value of 868.9336 within just 200 epochs, indicating the much higher convergence speed of the proposed Levenberg-Marquardt algorithm in comparison with the backpropagation algorithm. Furthermore, the sum square error plots in both Figure 6.7(a) and Figure 6.7(c) show that the training does not exhibit much oscillation. The results illustrated in Figure 6.7(b) and Figure 6.7(d) and also in Table 6.1(a) and Table 6.1(b) clearly show the excellent training and forecasting performance of the Takagi-Sugeno-type multiple-input, multiple-output neuro-fuzzy network with the proposed training algorithms.

Table 6.1(b). Training and forecasting performance of Takagi-Sugeno-type multi-input multi-output neuro-fuzzy network with proposed Levenberg-Marquardt algorithm for electrical load time series.

Sl. No.	Final SSE with pre-scaled data (scale factor = 0.001)	Final SSE, MSE, and RMSE with original (nonscaled) data
1.	SSE = 22.5777 SSE1 = 2.6365 SSE2 = 6.7828 SSE3 = 13.1584 (with training data 1 to 1500) (After training with Levenberg-Marquardt algorithm)	SSE = 2.2578e+005 MSE1 = 26.3650 MSE2= 67.8278 MSE3= 131.5837 RMSE1= 5.1347 RMSE2= 8.2358 RMSE3=11.471
2.	SSE = 42.3026 SSE1 = 5.0096 SSE2 = 11.7879 SSE3 = 25.5051 (with training and validation data points 1 to 3489)	SSE = 4.2303e+005 (with training and validation data points 1 to 3489)

Note that in the Table 6.1(a) and Table 6.1(b) SSE1, SSE2, and SSE3 indicate the sum squared error values at the output nodes 1, 2 and 3 respectively of the Takagi-Sugeno-type multi-input multi-output neuro-fuzzy network as formulated in (Equation 6.11a), and SSE indicates the cumulative sum of the sum square error values contributed by all three output nodes of the multi-input multi-output neuro-fuzzy network as formulated in (Equation 6.11b).

6.7.2 Prediction of Chaotic Time Series

In the next application example the proposed neuro-fuzzy algorithm has been tested for modelling and forecasting the Mackey-Glass chaotic time series, generated by solving the Mackey-Glass time delay differential equation (6.44) (MATLAB, 1998).

$$(dx/dt) = \left(0.2 \cdot x(t-\delta)\right/\left(1 + x^{10}(t-\delta)\right)) - 0.1 \cdot x(t), \tag{6.44}$$

for $x(0) = 1.2$, $\delta = 17$, and $x(t) = 0$, for $t < 0$.

The equation describes the arterial CO_2 concentration in the case of normal and abnormal respiration and belongs to a class of time-delayed differential equations that are capable of generating chaotic behaviour. It is a well-known benchmark problem in fuzzy logic and neural network research communities. Like in the

previous example for forecasting purposes the time series data $X = \{X_1, X_2, X_3, \ldots, X_q\}$ were rearranged in a multi-input single-output (XIO)-like structure. For modelling and forecasting of the given time series the respective neuro-fuzzy predictor that has to be developed is taken to have four inputs ($n = 4$) and one output ($m = 1$). In addition, both the sampling interval and the lead time of forecast is supposed to be six time units, so that for each $t > 18$ the input data represents a four-dimensional vector

$$\mathrm{XI}(t\text{-}18) = [X(t\text{-}18), X(t\text{-}12), X(t\text{-}6), X(t)],$$

and the output data a scalar value

$$\mathrm{XO}(t\text{-}18) = [X(t\text{+}6)].$$

In the forecasting example considered, using Equation (6.44) and neglecting the first 100 transient data from the chaotic series, in addition 1000 input-output data were generated for the XIO matrix. Out of 1000 generated input-output data, only the first 200 data sets were used for network training, and the remaining 800 data were used for verification of forecasting results.

The training and forecasting performances achieved with the implemented neuro-fuzzy network and with stored seven fuzzy rules are illustrated in Figure 6.8(a) and Figure 6.8(b) and listed in Table 6.2(a) and also compared with other standard models in Table 6.2(b). The items listed in serial numbers 1 to 12 of Table 6.2(b) were taken from Kim and Kim (1997), whereas serial number 13 is taken from Park *et al.* (1999). The results clearly confirm excellent training and forecasting performance of the Takagi-Sugeno-type neuro-fuzzy network for Mackey-Glass chaotic time series.

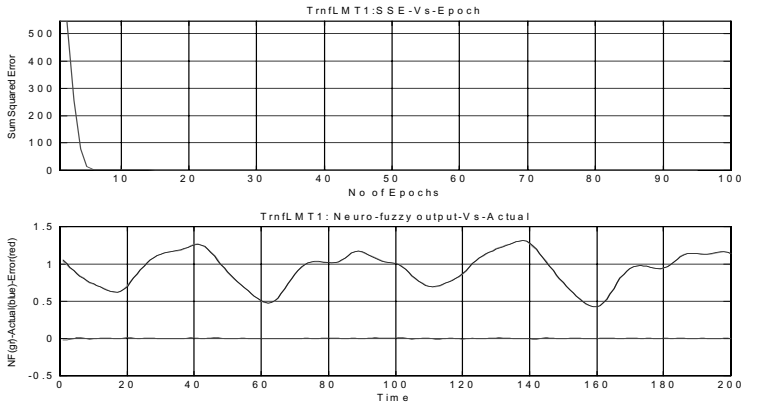

Figure 6.8(a). Training performance of Levenberg-Marquardt algorithm for Takagi-Sugeno-type of multi-input single-output neuro-fuzzy network (using seven fuzzy rules and seven GMFs) with Mackey-Glass chaotic time series data. Parameters of Levenberg-Marquardt algorithm: $\mu = 10$, $\gamma = 0.001$, $mo = 0.098$, $WF = 1.01$.

Table 6.2(a). Training and forecasting performance of Takagi-Sugeno-type of multi-input single-output neuro-fuzzy network (with $M = 7$ fuzzy rules) with proposed Levenberg-Marquardt algorithm for Mackey-Glass chaotic time series (SSE = sum square error, MSE = mean square error, MAE = mean absolute error, RMSE = root mean square error)

Sl. No.	Input data	SSE, MSE achieved	RMSE, MAE achieved
1.	1–200 (Training in 95 epochs)	SSE = 0.0026 MSE = 2.5571e–005	RMSE = 0.0051 MAE = 0.0039
2.	201–500 (Forecasting)	SSE = 0.0047 MSE = 3.1120e–005	RMSE = 0.0056 MAE = 0.0043
3.	501–1000 (Forecasting)	SSE = 0.0071	RMSE = 0.0053
4.	201–1000 (Forecasting)	SSE = 0.0118 MSE = 2.9427e–005	RMSE = 0.0054 MAE = 0.0042

SSE $= 0.5 \cdot \sum_{r=1}^{N} (e_r)^2$, MSE $= \sum_{r=1}^{N} (e_r)^2 \Big/ N$, RMSE $= \sqrt{\sum_{r=1}^{N} (e_r)^2 \Big/ N}$, and MAE $= \sum_{r=1}^{N} abs(e_r) \Big/ N$, where e_r is the error due to rth data sample and N is the number of data samples.

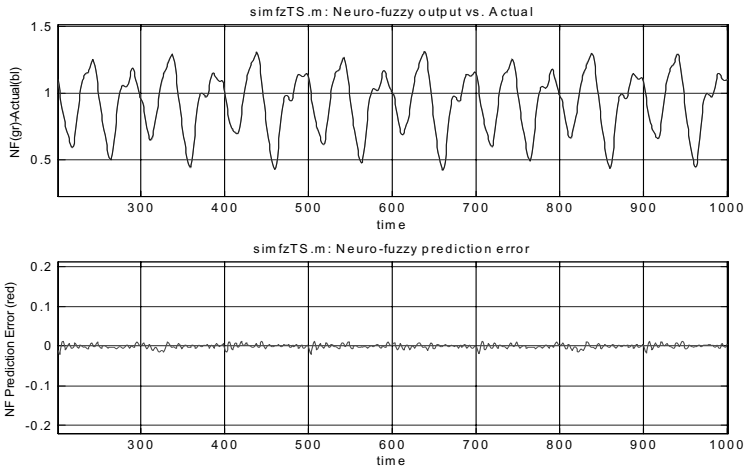

Figure 6.8(b). Performance of a Takagi-Sugeno-type multi-input single-output neuro-fuzzy network in forecasting the Mackey-Glass chaotic time series. Figure 6.8(a) and Figure 6.8(b) demonstrate the excellent training and forecasting performance of the Takagi-Sugeno-type multi-input single-output neuro-fuzzy network respectively for the Mackey-Glass chaotic time series. It is to be noted that the neuro-fuzzy network considered for this problem has only four inputs and one output and uses only seven Gaussian membership functions for (fuzzy) partitioning of input universes of discourse and seven fuzzy rules for neuro-fuzzy modelling.

Table 6.2(b). Comparison of training and prediction performance of fuzzy and other model with selected Takagi-Sugeno-type multi-input single-output neuro-fuzzy network with proposed Levenberg-Marquardt training algorithm for Mackey-Glass chaotic time series.

Sl. No.	Method	Training / forecasting with Input data	RMSE prediction error
1.	Kim and Kim (coarse partition)	500	0.050809 (5 partitions) 0.044957 (7 partitions) 0.038011 (9 partitions)
2.	Kim and Kim (after fine tuning)	500	0.049206 (5 partitions) 0.042275 (7 partitions) 0.037873 (9 partitions)
3.	Kim and Kim (genetic-fuzzy predictor ensemble)	500	0.026431
4.	Lee and Kim	500	0.0816
5.	Wang (product operator)	500	0.0907
6.	Min operator	500	0.0904
7.	Jang (ANFIS)	500	0.0070 (16 rules)
8.	Auto regression model	500	0.19
9.	Cascade correlation neural network	500	0.06
10.	Backpropagation neural network	500	0.02
11.	Sixth-order polynomial	500	0.04
12.	Linear prediction model	500	0.55
13.	FPNN (quadratic polynomial fuzzy inference)	500	0.0012 (16 rules)
14.	Takagi-Sugeno-type multi-input single-output neuro-fuzzy network (proposed work)	500 (forecasting)	0.0053 (7 rules, 7 GMFs, non-optimized)

RMSE = $\sqrt{\sum_{r=1}^{N}(e_r)^2 \Big/ N}$, e_r is the error due to rth training sample and N is the number of training / predicted data samples.

6.7.3 Modelling and Prediction of Wang Data

This example deals with the modelling of a second-order nonlinear plant

$$y(k) = g(y(k-1), y(k-2)) + u(k) \tag{6.45a}$$

studied by Wang and Yen (1998, 1999a, and 1999b) and by Setnes and Roubos (2000 and 2001), with

$$g(y(k-1), y(k-2)) = \frac{y(k-1)y(k-2)(y(k-1)-0.5)}{1+y^2(k-1)y^2(k-2)} \tag{6.45b}$$

The goal is to approximate the nonlinear component $g(y(k-1), y(k-2))$ of the plant with a suitable fuzzy model. Wang and Yen (1999) generated 400 simulated data points from the plant model (6.45a) and (6.45b). 200 samples of identification data were obtained with a random input signal $u(k)$ uniformly distributed in [-1.5, 1.5], followed by 200 samples of evaluation data obtained by using a sinusoid input signal $u(k) = \sin(2\pi k/25)$, as shown in Figure 6.9(a). This example was also used by Setnes and Roubos (2000 and 2001) and a comparison with the results of Wang and Yen (1998, 1999a, and 1999b) was made. Here, we also apply the proposed Takagi-Sugeno-type neuro-fuzzy modelling scheme on the original Wang-data and show the results for linear rules consequents and compare the results with others described in the above references.

In order to apply the Takagi-Sugeno-type neuro-fuzzy modelling scheme the original Wang data (which is available to us in the form of an XIO matrix of size 400 × 3 that contains the first two columns as inputs and the third column as the desired output) was scaled and normalized down to the range [0, 1] for convenience. In the following, since our objective is to approximate the nonlinear component $g(y(k-1), y(k-2))$ of the plant, the same is treated as the desired output from the neuro-fuzzy network, whereas $u(k)$ and $y(k)$ have been considered as two inputs to the neuro-fuzzy network. The scaling and normalization were performed separately on each column of the XIO matrix, *i.e.* $\text{XIO} = [u,\ y,\ g]$, and the three column vectors $u = [u_1, u_2, \cdots, u_N]^T$, $y = [y_1, y_2, \cdots, y_N]^T$ and $g = [g_1, g_2, \cdots, g_N]^T$, each contains N data points. The scaled and normalized vector

$$\begin{aligned} u_{\text{nsc}} &= K_0[(u_1 - u_{\min}), (u_2 - u_{\min}), \cdots, (u_N - u_{\min})]^T + u_{lo}, \\ K_0 &= (u_{hi} - u_{lo})/(u_{\max} - u_{\min}) \end{aligned} \tag{6.46}$$

is then computed where u_{max} and u_{min} are the maximum and minimum values of the u vector, and $u_{hi} = 1$ and $u_{lo} = 0$ are the desired highest and lowest values of the scaled or normalized u_{nsc} vector.

Once the scaling/normalization is performed, the scaled/normalized data are fed to the neuro-fuzzy network with $n = 2$ inputs and $m = 1$ output for training. Once

the network is trained, its final parameter values are stored and the network is used for prediction. In this experiment, the first 200 data samples were used for training and the remaining 200 data samples were used for evaluation. The training performance of the network is illustrated in Figure 6.9(b) and also listed in Table 6.3(a). It is also illustrated that using only $M = 10$ fuzzy rules (*first model*) and also 10 Gaussian membership functions implemented for fuzzy partition of the input universe of discourse the proposed training algorithm could bring the network performance index (SSE) down to 3.0836×10^{-4} or equivalently MSE to 3.0836×10^{-6} from their initial values 45.338 in only 999 epochs. This is equivalent to achieving an actual SSE = 0.0012 or an actual MSE = 1.1866×10^{-5} when computed back on the original data.

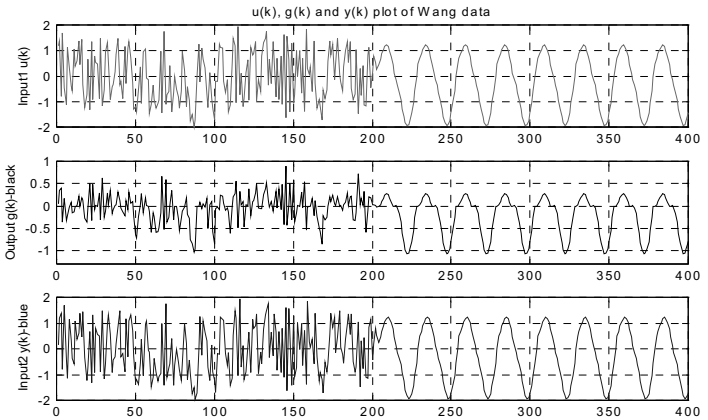

Figure 6.9(a). Plot of first input $u(k)$ (top), output $g(k)$ (middle) and second input $y(k)$ (bottom) of non-scaled Wang data (second-order nonlinear plant).

The corresponding evaluation performance of the trained network, as shown in Figure 6.9(c) and also listed in Table 6.3(a), illustrates that using the scaled or normalized evaluation data set from 201 to 400, the SSE value of 5.5319×10^{-4}, or equivalently MSE value of 5.5319×10^{-6}, were obtained. The above results further correspond to an actual SSE value of 0.0021, or equivalently to an actual MSE value of 2.1268×10^{-5}, which were computed back on the original evaluation data set. Evidently, the evaluation performance (actual MSE value), reported in Table 6.3(b), is at least 10 times better than that achieved by Setnes and Roubos (2000), Roubos and Setnes (2001) and much better than that achieved by Yen and Wang (1998, 1999a, 1999b).

Table 6.3(a). Training (999 epochs) and evaluation performance of Takagi-Sugeno-type multi-input single-output neuro-fuzzy network with proposed Levenberg-Marquardt algorithm for Wang data (second-order nonlinear plant data). Tuning parameter values of Levenberg-Marquardt algorithm for first model, *i.e.* $M = 10$, GMFs* = 10: $\mu = 10$, $\gamma = 0.01$, $mo = 0.098$, $WF = 1.05$; for second model, *i.e.* $M = 5$, GMFs* = 5: $\mu = 10$, $\gamma = 0.01$, $mo = 0.098$, $WF = 1.05$

Sl. No.	Input data	SSE & MSE (with pre-scaled and non-scaled actual data)	RMSE & MAE (with pre-scaled and non-scaled actual data)
1	1–200 Training data (*first model*)	SSE_train = 3.0836e–004 MSE_train = 3.0836e–006 Equivalently actual SSE_train = 0.0012 MSE_ train= 1.1866e–005	RMSE_train = 0.0018 MAE_train = 0.0012 Equivalently actual RMSE_train = 0.0034 MAE_train = 0.0024
2	201–400 Evaluation data (*first model*)	SSE_test = 5.5319e–004 MSE_test = 5.5319e–006 Equivalently actual SSE_test = 0.0021, MSE_test = 2.1268e–005	RMSE_test = 0.0024 MAE_test = 0.0015 Equivalently actual RMSE_test = 0.0046 MAE_test = 0.0030
3	1–200 Training data (*second model*)	SSE_train = 0.0135 MSE_train = 1.3491e–004 Equivalently actual SSE_train = 0.0519 MSE_train = 5.1866e–004	RMSE_train = 0.0116 MAE_train = 0.0087 Equivalently actual RMSE_train = 0.0228 MAE_train = 0.0170
4	201–400 Evaluation data (*second model*)	SSE_test = 0.0203 MSE_test = 2.0289e–004 Equivalently actual SSE_test = 0.0780 MSE_test = 7.8002e–004	RMSE_test = 0.0142 MAE_test = 0.0104 Equivalently actual RMSE_test = 0.0279 MAE_test = 0.0204

GMFs* = Gaussian membership functions

The same experiment was also carried out for $M = 5$ (*second model*), which exhibited the following training performance with the first 1 to 200 normalized and scaled training data: SSE and MSE values of 0.0135 and 1.3491×10^{-4} respectively, which correspond to the actual SSE and MSE values of 0.0519 and 5.1866×10^{-4} respectively. In addition, as listed below, the testing or evaluation performance of the Wang data with 201 to 400 rows, for five fuzzy rules and five Gaussian membership functions has produced SSE and MSE values of 0.0203 and 2.0289×10^{-4} respectively. These results further correspond to actual SSE and MSE values of 0.0780 and 7.8002×10^{-4} respectively, which are computed back from original (non-scaled) evaluation data.

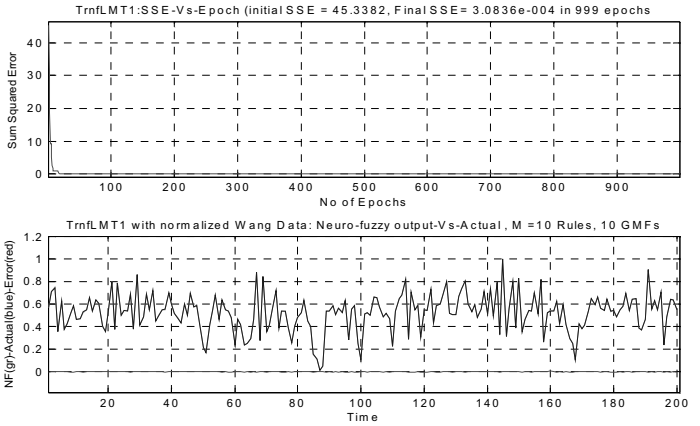

Figure 6.9(b). Performance of Takagi-Sugeno-type multi-input single-output neuro-fuzzy network with M = 10 rules (first model) for normalized Wang data when trained with proposed Levenberg-Marquardt algorithm.

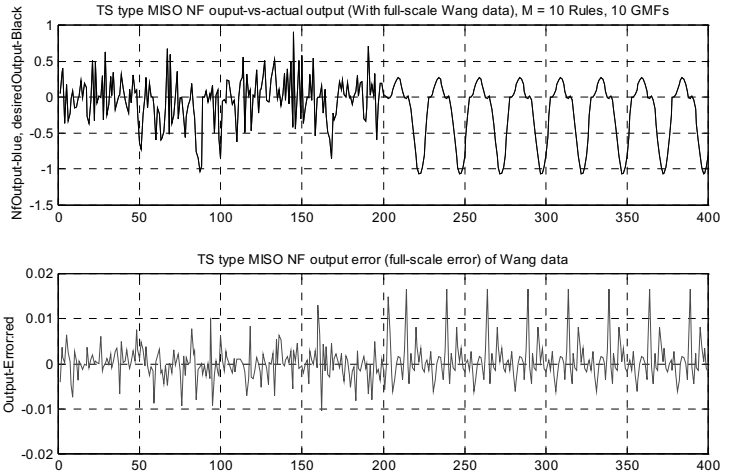

Figure 6.9(c). Prediction performance of Takagi-Sugeno-type multi-input single-output neuro-fuzzy network with M = 10 rules (first model) for non-scaled Wang data after training with proposed Levenberg-Marquardt algorithm.

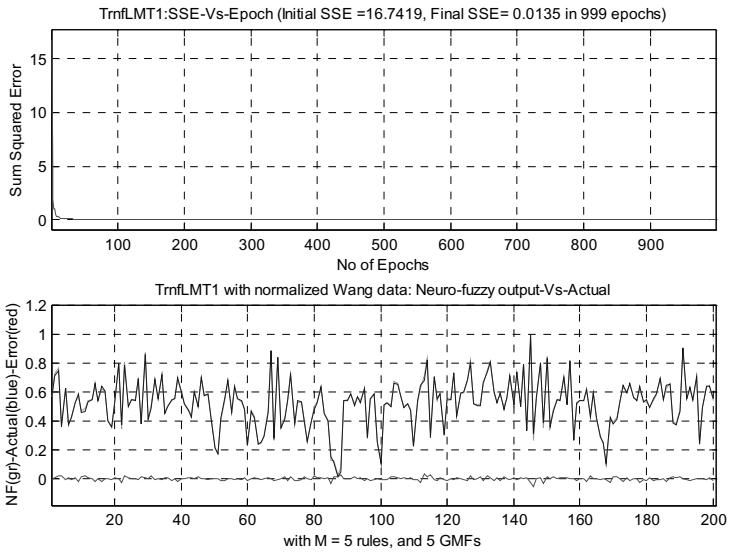

Figure 6.9(d). Performance of Takagi-Sugeno-type multi-input single-output neuro-fuzzy network with $M = 5$ rules (second model) and five GMFs for normalized Wang data when trained with proposed Levenberg-Marquardt algorithm.

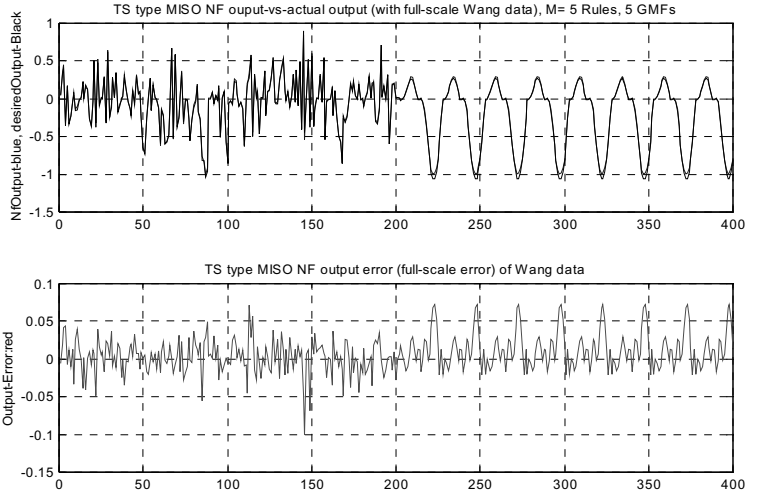

Figure 6.9(e). Prediction performance of Takagi-Sugeno-type multi-input single-output neuro-fuzzy network with $M = 5$ rules (second model) for non-scaled Wang data after training with proposed Levenberg-Marquardt algorithm.

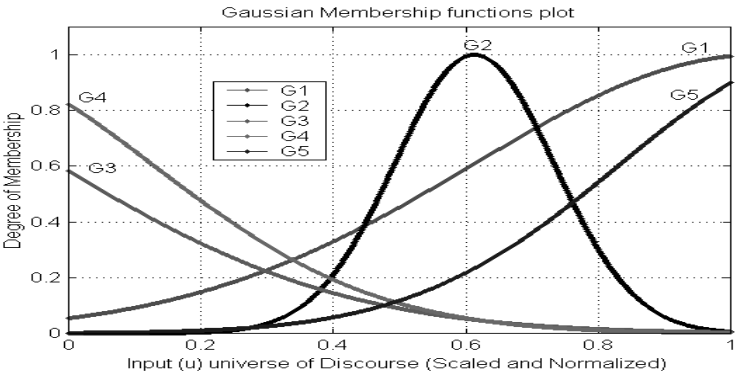

Figure 6.9(f). Finally tuned five Gaussian membership functions plot for fuzzy partition of input *u* (scaled/normalized) universe of discourse for Wang data. *X*-axis (input universe of discourse, scaled/normalized), *Y*-axis (degree of membership).

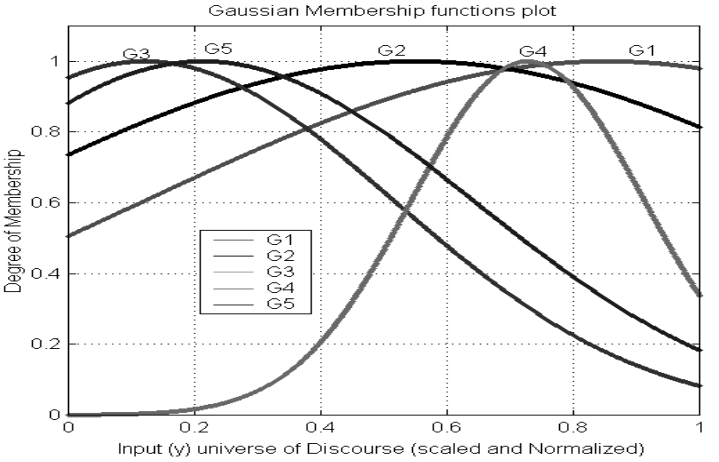

Figure 6.9(g). Finally tuned five Gaussian membership functions plot for fuzzy partition of input *y* (scaled/normalized) universe of discourse for Wang data. *X*-axis (input universe of discourse – scaled/normalized), *Y*-axis (degree of membership). Note that fuzzy membership functions in Figure 6.9(f) and Figure 6.9(g) are largely overlapping. Accuracy and transparency of the model are expected to be further improved if the similar fuzzy sets (for *e.g.* G4 and G3 in Figure 6.9(f) and in Figure 6.9(g) G3 and G5 are highly similar fuzzy sets) are merged and further fine-tuned using genetic algorithm or evolutionary computation.

Table 6.3(b). Comparison of training and evaluation performances of other fuzzy model and Takagi-Sugeno-type multi-input single-output neuro-fuzzy networks trained with the proposed Levenberg-Marquardt algorithm for Wang data (second-order nonlinear plant data)

Method	No. of rules	No. of fuzzy sets	Rules conseq.	MSE training	MSE eval.
Wang and Yen (1999)	40 (initial) 28 (optimized)	40 Gauss. (2D) 28 Gauss. (2D)	Singleton Singleton	3.3e–4 3.3e–4	6.9e–4 6.0e–4
Yen and Wang (1998)	36 (initial) 23 (optimized) 36 (initial) 24 (optimized)	12 B-splines 12 B-splines 12 B-splines 12 B-splines	Singleton Singleton Linear Linear	2.8e–5 3.2e–5 1.9e–6 2.0e–6	5.1e–3 1.9e–3 2.9e–3 6.4e–3
Yen and Wang (1999)	25 (initial) 20 (optimized)	25 Gauss. (2D) 20 Gauss. (2D)	Singleton Singleton	2.3e–4 6.8e–4	4.1e–4 2.4e–4
Setnes and Roubos (2000)	7 (initial) 7 (optimized) 5 (initial) 5 (optimized) 4 (optimized)	14 Triangular 14 Triangular 10 Triangular 8 Triangular 4 Triangular	Singleton Singleton Linear Linear Linear	1.6e–2 3.0e–3 5.8e–3 7.5e–4 1.2e–3	1.2e–3 4.9e–4 2.5e–3 3.5e–4 4.7e–4
Roubos and Setnes (2001)	5 (initial) 5 (optimized) 5 (optimized)	10 Triangular 10 Triangular 5 Triangular	Linear Linear Linear	4.9e–3 1.4e–3 8.3e–4	2.9e–3 5.9e–4 3.5e–4
Proposed neuro-fuzzy TS model	10 (initial, non-optimized) 5 (initial, non-optimized)	10 Gaussian 5 Gaussian	Linear Linear	1.1866e–5 5.1866e–4	2.1268e–5 7.8003e–4

The plots of the finally tuned GMFs that made the fuzzy partitions of universes of discourse of normalized input $u(k)$ and input $y(k)$ are shown in Figures 6.3(f) and 6.3(g) respectively. The figures also show that there is further scope for improving the accuracy, transparency and interpretability of neuro-fuzzy model obtained through similarity measures and genetic-algorithm-based optimizations. These issues, namely model transparency and interpretability, will be the main subject of discussion in Chapter 7. The results obtained in this example also, in general, summarize the excellent prediction performance of Takagi-Sugeno-type multi-input single-output neuro-fuzzy networks when trained with the proposed Levenberg-Marquardt Algorithm.

6.8 Other Engineering Application Examples

In the following, some engineering application examples are given in which the systematic neuro-fuzzy modelling approach has been used to solve the problem of

- material property prediction
- pyrometer reading correction in temperature measurement of wafers, based on prediction of wafer emissivity changes in a rapid thermal processing system, such as chemical vapour deposition and rapid thermal oxidation

- monitoring of tool wear.

6.8.1 Application of Neuro-fuzzy Modelling to Material Property Prediction

Chen and Linkens (2001) have proposed a systematic neuro-fuzzy modelling framework with application to mechanical property prediction in hot-rolled steel. Their methodology includes three main phases:

- the initial fuzzy model, which consists of generation of fuzzy rules by a self-organizing network
- the second phase, which includes the selection of important input variables on the basis of the initial fuzzy model and also the assessment of the optimum number of fuzzy rules (hidden neurons in the RBF network) and the corresponding receptive fields determination via the fuzzy *c*-means clustering algorithm
- third phase, dedicated to the model optimization, including parameter learning and structure simplification on the basis of backpropagation learning and the similarity analysis of fuzzy membership functions.

Thereafter, the neuro-fuzzy model developed is used to predict the tensile stress, yield stress, and the like in materials engineering.

In materials engineering, property prediction models for materials are important for design and development. This has for many years been an important subject of research for steel. Much of this work has concentrated on the generation of structure - property relationships based on linear regression models (Pickering, 1978), (Hodgson, 1996), developed only for some specific class of steels and specific processing routes. Recently, some improved, neural-networks-based models have been developed for prediction of mechanical properties of hot-rolled steels (Hodgson, 1996), (Chen *et al.*, 1998), and (Bakshi and Chatterjee, 1998). Using complex nonlinear mapping, the models provide more accurate prediction than traditional linear regression models. But the drawback here is that the development of these kinds of model is usually highly problem specific and time consuming, so that the development of a fast, efficient, and systematic data-driven modelling framework for material property prediction is still needed.

The problem of modelling of hot-rolled metal materials can be broadly stated as follows. Given a certain material which undergoes a specified set of manufacturing processes, what are the final properties of this material? Typical final properties, in which metallurgical engineers are interested, are the mechanical properties such as, tensile strength (*TS*), yield stress, elongation, *etc.* Chen *et al*. (2001) have developed a neuro-fuzzy model for the prediction of the composition-microstructure-property relations of a wide range of hot-rolled steels. More than 600 experimental data from carbon-manganese (C-Mn) steels and niobium micro-alloyed steels have been used to train and test the neuro-fuzzy model, which relates the chemical compositions and microstructure with the mechanical properties.

In the experimental data set, they have considered 13 chemical compositions, two microstructure variables, and measured tensile stress values, which corresponds to a system with 15 possible input variables and with one output

variable. Several performance indices (RMSE and MAE), and the correlation coefficient between the measured and the model predicted tensile stress were used to evaluate the performance of the fuzzy models developed. Property prediction results for different types of steel are summarized below.

6.8.1.1 Property Prediction for C-Mn Steels

Using the proposed input selection paradigm, five inputs (the carbon, silicon, manganese, nitrogen contents and the ferrite grain size $D^{-1/2}$ ($mm^{-1/2}$), were selected from the 15 possible input variables. Three hundred and fifty-eight industrial data were used, with 50% of them for training and the remaining 50% for model testing. After partition validation and parameter learning, the final fuzzy models of the Mamdani type consisting of six rules were obtained. The rule-based fuzzy model was represented by six fuzzy rules. From the fuzzy model generated, Chen *et al.* (2001) used linguistic hedges to derive the corresponding linguistic model.

The fuzzy model with linguistic hedges finally generated used six Mamdani-type fuzzy rules, such as one described below:

Rule-1: IF Carbon is large and Silicon is medium and Manganese is large and Nitrogen is medium and $D^{-1/2}$ is more or less medium, THEN Tensile Stress is large

Using the above model, Chen *et al.* (2001) obtained good prediction results that gave RMSE = 12.44 and 16.85 and MAE = 9.46 and 13.15 for model training and testing respectively.

According to their simulation result, the out-of-10% error-band prediction patterns for the testing data is 2.2%. It was claimed that the fuzzy model generated gave good prediction and generalization capability.

6.8.1.2 Property Prediction for C-Mn-Nb Steels

In another experiment of Chen *et al.* (2001), for property prediction for C-Mn-Nb steels, more than 600 measured data, including the previously used 358 C-Mn data, were used to build the fuzzy model. Three hundred and fifteen data were selected for training and the remaining 314 data were used for testing. Using their proposed fuzzy modelling approach, six out of 15 variables were selected as the inputs (C, Si, Mn, N, Nb, $D^{-1/2}$) with tensile stress as output. A six-rule fuzzy model was developed after structure identification and parameter training. The property prediction resulted in RMSE = 15.48 and 19.74 and MAE = 12.11 and 14.46 for training and testing, respectively. Furthermore, the out-of-10% error-band patterns for the testing data were found to be only 3%.

6.8.2 Correction of Pyrometer Reading

As a second engineering application, we describe here the prediction capability of a self-constructing ***neural-fuzzy inference network*** (SONFIN) proposed by Lai and Lin (1999) for pyrometer reading correction in wafer temperature measurement, based on emissivity changes. The motivation for this was that,

because of several distinct advantages of rapid thermal processing (RTP) over other batch processing, such as significant reduction in thermal budget and better control over the processing environment, rapid thermal processing has been extensively used in high-density integrated circuit manufacturing on single wafers.

Wafer temperature measurement and control are two critical issues here. Currently, a single-wavelength pyrometer is used as a non-contact temperature sensor. However, for applications where the characteristics of the surface change with the time, the wafer emissivity also varies simultaneously. This can lead to temperature errors in excess of 50 degree Celsius in a few seconds. Various methods were suggested to overcome this problem, such as use of a dual-wavelength pyrometer, model-based emissivity correction, *etc*. A global mathematical model for the rapid thermal process, which includes the temperature sensor along with a control loop and lamp system, was developed and simulated by Lai and Lin (1999). In the same model, emissivity changes during oxidation are calculated according to reflections, refraction within thin dielectric films on a silicon substrate. The oxide thickness as a function of oxidation time at various temperatures, is simulated by a linear parabolic model. Using the basic heat transfer law, a pyrometer model to simulate the temperature sensor in the rapid thermal process is derived and, thereafter, a neural-fuzzy network is used to learn and predict the variations of oxidation growth rate of the film under different process temperatures. Based on this neural-fuzzy prediction and an already available optical model the emissivity of the wafer can be correctly computed.

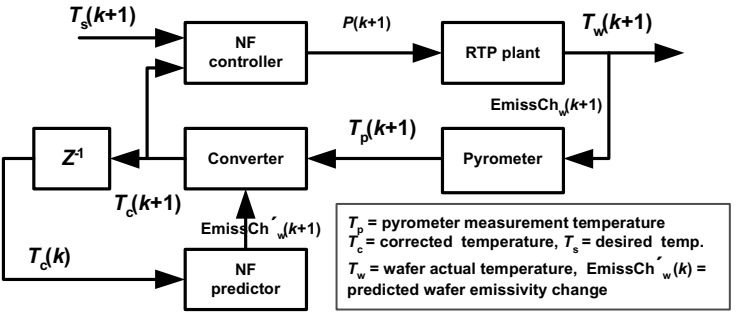

Figure 6.10. Block diagram of the neural-fuzzy method to predict wafer emissivity variation and to correct the pyrometer readings

Another neural-fuzzy network was used by Lai and Lin (1999) to control the temperature of an RTP system by using the inverse model of the RTP system to achieve two control objectives: trajectory following and temperature uniformity on the wafer. Figure 6.10 shows the block diagram of the neural-fuzzy method to predict wafer emissivity variation and to correct the pyrometer readings. The previous corrected temperature value $T_c(k)$ and the current processing time k are used as the inputs of the neural-fuzzy network to predict the current film thickness, which is further used to compute the emissivity of the wafer $ew'(k+1)$ according to

wafer optical model. The converter is then used to correct the pyrometer reading value $T_p(k+1)$ to $T_c(k+1)$.

The neural-fuzzy network used for this purpose was the SONFIN, which has a fuzzy rule-based network possessing neural learning ability. Compared with other existing neural-fuzzy networks, a major characteristic of this network is that no preassignment and design of fuzzy rules are required. The rules are constructed automatically during the training process. Besides, SONFIN can overcome both the difficulty of finding a number of proper rules for the fuzzy logic controllers and the overtuned and slow convergence phenomena of backpropagation neural networks. SONFIN can also optimally determine the consequent of fuzzy IF-THEN rules during its structure learning phase, and it also outperforms the pure neural networks greatly, both in learning speed and accuracy.

6.8.3 Application for Tool Wear Monitoring

In automated manufacturing systems, such as flexible manufacturing systems, one of the most important issues is the detection of tool wear during the cutting process to avoid poor quality in the product or even damage to the workpiece or the machine. It will be shown that a neuro-fuzzy model, based on a prediction technique, can be applied for monitoring tool wear in the drilling process.

The alternating direction of the cutting force leads to vibrations of the machine structure. These vibrations will change owing to the tool wear conditions. Despite the relatively harsh environment in the proximity of the cutting zone, the vibrations can be measured conveniently by accelerometers at a comparably affordable price. Neural networks have, for a long time, been used for classification of various signals. However, because of many limitations, including the slow training performance of neural networks, alternatively a neural network with fuzzy inference has been used because of its much faster learning ability. The latter is nothing but a neuro-fuzzy type of hybrid learning network. Using such a network a new drill condition monitoring method is described, as proposed by Li *et al.* (2000). The method is based on spectral analysis of the vibration signal. The results are used to generate a set of **indices for monitoring**, utilizing the fact that the frequency distribution of vibration changes as the tool wears. The nonlinear relationship between the tool wear condition and these monitoring indices is modelled using a hybrid neuro-fuzzy network. The hybrid network selected in this case has five inputs and five outputs. The inputs to the network are the monitoring indices based on the vibration signal of the drilling process. It is to be noted that the mean value of each frequency band can be used to characterize the different tool conditions. The monitoring indices selected as network inputs are summarized in Table 6.4(a). The content of the Table 6.4(a) is read follows:

x_1 = the r.m.s value of the signal in the frequency band [0, 300] Hz.

Unlike the inputs of the network, the tool wear condition of the network was divided into five states represented by five fuzzy membership functions (MF), namely initial wear, normal wear, acceptable wear, severe wear and failure. Based on the flank wear of the tool, these conditions are summarized in the Table 6.4(b).

Table 6.4(a). Summary of monitoring indices selected as network inputs

Input terminal of network	Input representation	RMS value of the signal in the frequency band
1	x_1	[0, 300] Hz.
2	x_2	[300, 600] Hz
3	x_3	[600, 1000] Hz
4	x_4	[1000, 1500] Hz
5	x_5	[1500, 2500] Hz

Table 6.4(b). Summary of the conditions for various flank wear

Output terminal of the network	Fuzzy MF	Tool condition	Flank wear
y_1	1	Initial wear	0 < wear < 0.1 mm
y_2	2	Normal wear	0.05 < wear < 0.3 mm
y_3	3	Acceptable wear	0.25 < wear <0.5 mm
y_4	4	Severe wear	0.45< wear <0.6 mm
y_5	5	Failure	wear > 0.6 mm

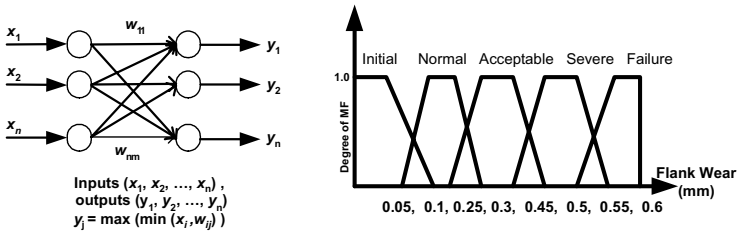

Figure 6.11. Fuzzy neural net topology (left), fuzzy membership functions of drilling conditions (right)

The fuzzy membership functions of drilling conditions based on experimental data and the observed system behaviour are set for output indices of the hybrid network, and are shown in the Figure 6.11. The reason for choosing a trapezoidal membership function is that it is difficult to quantify what exact percentage of tool wear corresponds to a certain linguistic variable. In order to improve the training speed of the hybrid network, the tool wear conditions are coded as follows: initial (1,0,0,0,0); normal (0,1,0,0,0); acceptable (0,0,1,0,0); severe (0,0,0,1,0); and failure

(0,0,0,0,1). If the tool condition is acceptable, then the output values of the hybrid network are (0,0,1,0,0).

Once the hybrid neuro-fuzzy network has learnt the above nonlinear mapping from a given set of training examples consisting of (x, y) values, thereafter, for a new set of monitoring indices (*i.e.* related to the frequency band of the vibration), obtained from the drilling process through accelerometer, charge amplifier, and the signal processing unit, the network will generate or predict a set of y values. The maximum of y_i, namely J, is converted to 1, and the others are converted to 0. For instance, if $J = \max\{y_i \mid i = 1, 2, 3, \ldots, 5) = y_2 = 0.8$, the predicted output of the hybrid network is (0,1,0,0,0). This prediction indicates that the tool wear condition belongs to the normal category. Exploiting the prediction capability of the hybrid network and adopting similar methodologies one can monitor the tool wear in an automated manufacturing system.

6.9 Concluding Remarks

In this chapter a hybrid neuro-fuzzy modelling frame work is proposed. An accelerated training algorithm, based on either the backpropagation or Levenberg-Marquardt algorithm and in combination with a modified error index extension, has also been developed for training Takagi-Sugeno-type multi-input multi-output or multi-input single-output neuro-fuzzy networks. The increased speed of training convergence was experimentally confirmed on some examples of modelling and forecasting of time series. It was observed that the addition of a small modified error index term to the original performance function improves the convergence speed of both standard backpropagation and the Levenberg-Marquardt algorithm significantly.

The trained neuro-fuzzy network itself is found to be powerful for modelling and prediction of dynamics of various nonlinear phenomena. However, the fuzzy rules generated through neuro-fuzzy training are occasionally found to be not transparent enough, in the sense that a clear interpretation of all the tuned fuzzy sets is not possible. This is due to the fact that the membership functions, finally tuned through neuro-fuzzy network training, are frequently very similar to each other or they greatly overlap each other, giving rise to a difficult situation to interpret. To solve this problem and to improve the interpretability of fuzzy rules, set-theoretical ***similarity measures*** should be computed for each pair of fuzzy sets and highly similar fuzzy sets should be merged together into a single set (Setnes, Babuška, Kaymark, 1998) as discussed in detail in Chapter 7. Furthermore, the tuned membership functions building a universal fuzzy set within the universe of discourse should be removed because they do not contribute anything to the rule base. Also, because the parameters of the Gaussian membership functions are unconstrained, it is probable that the fuzzy partition occasionally may not look like the usual fuzzy partition. In such cases, the interpretation of a trained neuro-fuzzy system may also not be possible.

An additional issue is the determination of the optimum number of fuzzy rules and hence, also the determination of optimum number of membership functions. This is essential, because an unnecessarily larger rule base may overfit the noisy

data and thereby worsen the prediction ability. For determination of the optimum number of rules and of membership functions, genetic algorithms or, in general, evolutionary computation, should preferably be used as a proper support tool.

It should finally be underlined that, after the completion of backpropagation or Levenberg-Marquardt training, if the final (linear/singleton) rules consequent parameters are determined by applying the least squares error estimator using only the tuned GMF parameters of the network, then the accuracy of the model could occasionally be increased further. Furthermore, the simulation results have shown that the Levenberg-Marquardt algorithm, based on Jacobian matrices computed using normalized prediction error or normalized equivalent error (Section 6.4.2.3.1), though computationally very heavy, often leads to a better training performance and to a faster convergence when applied to the Takagi-Sugeno type of neuro-fuzzy networks. In the experiments investigated here, the proposed training algorithms (modified backpropagation/Levenberg-Marquardt algorithm) proved to be efficient enough for neuro-fuzzy modelling and for prediction of electrical load time series, chaotic time series, *etc*. Furthermore, some recently published additional engineering examples confirm the versatility and possible other applications of neuro-fuzzy networks in different fields of engineering.

References

[1] Bakshi BR and Chatterjee R (1998) Unification of neural and statistical methods as applied to materials structure-property mapping, J. Alloys Compounds, 279(1): 39–46.

[2] Bezdek JC (1993) Editorial-fuzzy models: What are they and why, IEEE Trans. on Fuzzy Systems, vol. 1, pp. 1–5.

[3] Brown M and Harris C (1994) Neuro-fuzzy adaptive modelling and control, Prentice Hall, New York.

[4] Buckley JJ and Hayashi Y (1994) Fuzzy neural networks, In: Fuzzy Sets, Neural Networks and Soft Computing, edited by Yager R and Zadeh L, Van Nostrand Reinhold, New York.

[5] Chak CK, Feng G and Ma J (1998) An adaptive fuzzy-neural network for MIMO system model approximation in high-dimensional spaces, IEEE Trans. on System, Man and Cybernetics, 28: 436–446.

[6] Chen M and Linkens DA (1998) A fast fuzzy modeling approach using clustering neural networks, In Proc. IEEE world congress on Intell. Computat. 2: 1406–1411.

[7] Chen M Linkens DA (2001), A systematic neuro-fuzzy modelling framework with application to material property prediction, IEEE Trans. on SMC, B 31(5): 781–790.

[8] Cho KB and Wang BH (1996) Radial basis function based adaptive fuzzy systems and their application to system identification and prediction, Fuzzy Sets System., 83: 325–339.

[9] Fuller R (1995) Neural-fuzzy systems, Abo Akademi.

[10] Gupta MM (1994) Fuzzy neural networks: Theory and Applications, Proceedings of SPIE, vol. 2353, pp. 303–325.

[11] Gustafson DE, Kessel WC (1979) Fuzzy clustering with fuzzy covariance matrix, Proc. of the IEEE CDC, San Diego, 761–766.

[12] Hagan MT, Menhaj MB (1994) Training feedforward networks with the Marquardt algorithm, IEEE Trans. on Neural Networks, 5(6): 989–993.

[13] Hodgson PD (1996) Microstructure modeling for property prediction and control, J. of Materials Process Technology, 60: 27–33.
[14] Jang JSR (1993) ANFIS: Adaptive Network Based Fuzzy Inference System, IEEE Trans. on SMC., 23(3): 665–685
[15] Jang JSR, Sun CT (1995) Neuro-fuzzy modelling and control, Proc. of IEEE, 83: 378–406.
[16] Kim D, Kim C (1997) Forecasting time series with genetic fuzzy predictor ensemble, IEEE Trans. on Fuzzy Systems, 5(4): 523–535.
[17] Kosko B (1992) Neural networks and fuzzy systems, Prentice Hall, Englewood Cliffs, New Jersey.
[18] Kulkarni AD (1998) Neural-fuzzy models for multi-spectral image analysis, Internat. J. of Applied Intelligence, 8: 173–187
[19] Kulkarni AD (2001) Computer vision and fuzzy-neural systems, Upper Saddle River, New Jersey: Prentice Hall PTR.
[20] Lai JH and Lin CT (1999) Application of neural fuzzy network to pyrometer correction and temperature control in rapid thermal processing, IEEE Trans. Fuzzy Systems, 7(2):160–174.
[21] Lee SH, Kim I (1994) Time series analysis using fuzzy learning, Proc. of Intern. Conf. on Neural Information Processing, Seoul, Korea, 6: 1577–1582.
[22] Li X, Dong S, Venuvinod PK (2000) Hybrid Learning for tool wear monitoring, Int. J. Adv. Manufacturing Technology, 16: 303–307.
[23] Lin CT and Lee CSG (1991) Neural networks based fuzzy logic and control systems, IEEE Trans. On Computers, vol. 40, pp. 1320–1336.
[24] MATLAB (1998) Fuzzy logic toolbox, user's guide, The Math Works Inc., vers. 5.2
[25] Mitra S, Hayashi Y (2000) Neuro-fuzzy rule generation: survey in soft computing framework, IEEE Trans. on Neural Networks, 11(3): 748–768.
[26] Nauck D, Klawonn F and Kruse R (1997) Foundations of neuro-fuzzy systems, Wiley, Chichester, U.K.
[27] Nie J (1997) Nonlinear time-series forecasting : A fuzzy-neural approach, Neurocomputing, 16(1997): 63–76.
[28] Pal SK and Mitra S (1992) Multilayer perceptron, fuzzy sets and classification, IEEE Trans. On Neural Networks, 2(5): 683–697.
[29] Palit AK and Babuška R (2001) Efficient training algorithm for Takagi-Sugeno type neuro-fuzzy network, Proc. of FUZZ-IEEE, Melbourne, Australia, vol. 3: 1367–1371.
[30] Palit AK and Popovic D (1999) Forecasting chaotic time series using neuro-fuzzy approach, Proc. of IEEE-IJCNN, Washington DC, USA, vol. 3: 1538–1543.
[31] Palit AK and Popovic D (1999) Fuzzy logic based automatic rule generation and forecasting of time series, Proc. of FUZZ-IEEE, Seoul, Korea, vol. 1: 360–365.
[32] Palit AK and Popovic D (2000) Intelligent processing of time series using neuro-fuzzy adaptive genetic approach, Proc. of IEEE-ICIT, Goa, India, vol. 1:141–146.
[33] Palit AK and Popovic D (2000) Nonlinear combination of forecasts using artificial neural network, fuzzy logic and neuro-fuzzy Approaches, Proc. of FUZZ-IEEE, San Antonio, Texas, USA, vol. 2: 566–571.
[34] Palit AK, Doeding G, Anheier W, Popovic D (2002) Backpropagation based training algorithm for Takagi-Sugeno type MIMO neuro-fuzzy network to forecast electrical load time series, Proc. of FUZZ-IEEE, Honolulu, Hawai, USA. vol. 1: 86–91.
[35] Park HS, Oh SK, Ahn TC and Pedrycz W (1999) A study on multi-layer based fuzzy polynomial inference system based on an extended GMDH algorithm, Proc. of FUZZ-IEEE, Seoul, Korea, vol. 1: 354–359
[36] Pedrycz W (1995) Fuzzy sets engineering, CRC Press, Boca Raton, Florida.
[37] Pickering FB (1978) Physical metallurgy and the design of steels, Applied Science, London, U.K.

[38] Roubos H, Setnes M (2001) Compact and transparent fuzzy models and classifiers through iterative complexity reduction, IEEE Trans. on Fuzzy System, 9(4): 516–524
[39] Setnes M, Babuška R, Kaymark U, *et al.*, (1998) Similarity measures in fuzzy rule base simplification, IEEE trans. on SMC., B-28: 376–386
[40] Setnes M, Roubos JA (2000) GA-fuzzy modelling and classification: complexity and performance, IEEE Trans. on Fuzzy Systems, 8(5): 509–522
[41] Takagi and Hayashi (1991) NN-driven fuzzy reasoning, Internat. J. of Approximate Reasoning, 5(3): 191–212.
[42] Wang L and Yen J (1999) Extracting fuzzy rules for system modelling using a hybrid of genetic algorithms and Kalman filter, Fuzzy Sets System, 101: 353–362
[43] Wang LX (1994) Adaptive fuzzy systems and control: design and stability analysis, Englewood Cliffs, New Jersey: Prentice Hall.
[44] Wang LX and Mendel JM (1992a) Fuzzy basis functions, universal approximation, and orthogonal least squares learning, IEEE Trans. on Neural Network, 3: 807 – 814.
[45] Wang LX and Mendel JM (1992b) Back-propagation fuzzy system as nonlinear dynamic system identifiers, Proc. of FUZZ-IEEE, vol. 2: 1409–1418.
[46] Wang LX and Mendel JM (1992c) Generating fuzzy rules by learning from examples, IEEE Trans. on SMC, 22(6): 1414–1427.
[47] Xiaosong D, Popovic D, Schulz-Ekloff G (1995) Oscillation resisting in the learning of backpropagation neural networks, Proc. of 3rd IFAC/IFIP workshop on algorithm and architectures for real-time control, Ostend, Belgium.
[48] Yen J and Wang L (1998) Application of statistical information criteria for optimal fuzzy model construction, IEEE Trans. on Fuzzy System, 6(3): 362–371.
[49] Yen J and Wang L (1999) Simplifying fuzzy rule-based models using orthogonal transformation methods, IEEE Trans. on SMC, 29(1): 13–24.
[50] Zhang J and Morris AJ (1999) Recurrent neuro-fuzzy networks for nonlinear process modelling, IEEE Trans. on Neural Networks, 10: 313–326.