

TAMPERE UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING

TOMMI OJALA

NEURO-FUZZY SYSTEMS IN CONTROL

MASTER OF SCIENCE THESIS

SUBJECT APPROVED BY DEPARTMENTAL
COUNCIL ON JUNE 8, 1994

Examiners: Assistant Prof. Petri Vuorimaa
M.Sc. Vesa Ruoppila

Address	Telephone	Telefax	Telex
P.O. Box 692, FIN-33101 Tampere, Finland	+358 31 316 2111	+358 31 316 2160	22313 ttkr fi

Preface

This M.Sc. thesis has been done at Signal Processing Laboratory, Tampere University of Technology. The work was financed by Finnish Academy and Tampere University of Technology.

I would like to thank my thesis examiners Assistant Prof. Petri Vuorimaa and M.Sc. Vesa Ruoppila for guidance and support. I would also like to thank Mr. Tarmo Jukarainen, M.Sc., Mr. Mika Tervonen, and the whole staff of the laboratory for several useful discussions and excellent working atmosphere. My warmest thanks to my dear wife Riitta for her support, patience, and love.

Tampere, January 4, 1995

Tommi Ojala
Pellavatehtaankatu 19 A 1
33100 TAMPERE
tel. (931) 2121 389

Contents

Abstract	4
Tiivistelmä	5
List of abbreviations	6
List of symbols	7
1 Introduction	9
2 Basics of neural networks	11
2.1 Biological and artificial neural networks	11
2.2 Radial basis function network	14
2.3 Self-organizing map	17
3 Basics of fuzzy logic	23
3.1 Fuzzy sets	23
3.2 Fuzzy reasoning	25
4 Neuro-fuzzy systems	29
4.1 Background of neuro-fuzzy systems	29
4.2 Neural fuzzy inference systems	31
5 Fuzzy self-organizing map	40
5.1 Basic structure	40
5.2 Learning	44
5.3 Generation of fuzzy rules	47
5.4 FSOM using Sugeno's fuzzy rules	51
5.5 FSOM simulation studies	54
6 Simulation examples	58
6.1 Identification of a heating process	58
6.2 Model-based neuro-fuzzy controller	63
6.3 Fault diagnosis of a reactor and a heat exchanger system	71
7 Discussion	76
8 Summary	78
References	80

TAMPERE UNIVERSITY OF TECHNOLOGY

Degree program in automation engineering

Signal Processing Laboratory

Ojala, Tommi: Neuro-fuzzy Systems in Control

Master of Science Thesis, 85 p.

Examiners: Assistant Prof. Petri Vuorimaa, M.Sc. Vesa Ruoppila

Funding: Finnish Academy and Tampere University of Technology

January 1995

Neuro-fuzzy systems combine the theory of two popular computational intelligence techniques: neural networks and fuzzy logic. Research on neuro-fuzzy systems is still young, but several systems have already been proposed. In this thesis, a comprehensive literature survey of the neuro-fuzzy systems is given. The main interest is focused on the systems which are capable of extracting fuzzy knowledge by using learning methods suggested for neural networks.

The main purpose of this thesis is to examine a neuro-fuzzy system called fuzzy self-organizing map (FSOM). The FSOM is a fuzzy version of Kohonen's self-organizing map neural network model. In the FSOM, neurons of the original self-organizing map are replaced by fuzzy rules and fuzzy sets. In contrast to many other neuro-fuzzy systems, the learning of the system is based on a modified Kohonen's Learning Vector Quantization. The FSOM can be used as neuro-fuzzy classifier, but also as neuro-fuzzy approximator. The learning of the FSOM and an alternative fuzzy set operation are studied using computer simulations. In addition, several proposals for future improvement of the system are presented. In this thesis a novel version of the FSOM is proposed. It is more capable of modelling nonlinear systems than the original FSOM, since the behaviour of the system can be stated as a combination of local linear models.

The other major interest of this work is to examine applications of neuro-fuzzy systems to system identification, control design, and fault diagnosis. The results show that the FSOM is the most plausible in the fault diagnosis problem. In the identification example, the best results are reached by the FSOM version which is developed in this thesis. The FSOM is a potential tool, although the development of the algorithms is still necessary.

TAMPEREEN TEKNILLINEN KORKEAKOULU

Automaatiotekniikan koulutusohjelma

Signaalinkäsittelyn laitos

Ojala, Tommi: Neuro-sumeat järjestelmät säädössä

Diplomityö, 85 s.

Tarkastajat: yliass. Petri Vuorimaa ja DI Vesa Ruoppila

Rahoittaja: Suomen Akatemia ja Tampereen teknillinen korkeakoulu

Tammikuu 1995

Neuro-sumeissa järjestelmissä yhdistyvät kaksi laskennallisen älykkyyden osa-aluetta: neuraaliverkot ja sumea logiikka. Vaikka neuro-sumeiden järjestelmien tutkimus on vielä nuorta, useita järjestelmiä on jo kehitetty. Tässä työssä kiinnostuksen kohteena ovat neuro-sumeat järjestelmät, jotka kykenevät neuraaliverkoista tuttuja oppimismenetelmiä käyttäen automaattisesti luomaan ja virittämään sumeita sääntöjä ja jäsenyysfunktioita.

Työssä tutkitaan sumeaa itseorganisoituvaa karttaa, joka on sumea versio Kohosen itseorganisoituvasta kartasta. Sumeassa itseorganisoituvassa kartassa alkuperäisen mallin neuronit on korvattu sumeilla säännöillä. Useista muista järjestelmistä poiketen sumean itseorganisoituvan kartan opetus perustuu muunnettuun versioon oppivasta vektorikvantisoinnista (LVQ). Sumeaa kartta voidaan myös alkuperäisestä Kohosen mallista poiketen käyttää epälineaaristen järjestelmien mallintamisessa. Sumean itseorganisoituvan kartan oppimista sekä vaihtoehtoista sumeiden joukkojen yhdistämisessä käytettävää päättelyoperaattoria tutkitaan tietokonesimuloinnein. Työssä esitetään useita parannusehdotuksia sumean itseorganisoituvan kartan kehittämiseksi. Samoin esitellään myös uudentyypinen sumea itseorganisoituva kartta, joka soveltuu alkuperäistä sumeaa mallia paremmin sellaisten epälineaaristen prosessien mallinnukseen, joiden toiminta voidaan kuvata paikallisilla lineaarisilla malleilla.

Työn päätavoitteena oli tutkia ja kehittää menetelmiä sumean itseorganisoituvan kartan käyttämiseksi säätösovelluksissa. Sumean kartan käyttö esitetään kolmessa esimerkkitapauksessa, jotka edustavat identifioinnin, säätäjäsuunnittelun ja vikadiagnostiikan ongelmia. Simulointitulosten perusteella sumea itseorganisoituva kartta sopii erityisen hyvin vikadiagnostiikkaan. Identifioinnissa parhaimmat tulokset saavutetaan tässä työssä kehitetyllä sumean itseorganisoituvan kartan mallilla. Sumea itseorganisoituva kartta on käytökelpoinen neuro-sumea malli, vaikkakin sen jatkokehitys vaikeampia ongelmia silmällä pitäen onkin tarpeen.

List of abbreviations

ARX	AutoRegressive with eXogenous input
COA	Center Of Area
FIS	Fuzzy Inference System
FSOM	Fuzzy Self-Organizing Map
LMS	Least Mean Squares
LSE	Least Squares Estimate
LVQ	Learning Vector Quantization
MLP	MultiLayer Perceptron
OLS	Orthogonal Least Squares
PR	Pseudo Random
RBF	Radial Basis Function
RCE	Restricted Coulomb Energy
RMS	Root Mean Squares
SOM	Self-Organizing Map
WA	Weighted Average

List of symbols

Radial basis function network

c_i	Center of the i th receptive field unit
f_j	j th output of the network
σ_i	Width of the i th Gaussian exponent function
R_i	Output of the i th receptive field unit
\mathbf{x}	Input vector
$w_{i,j}$	Output weight of the i th basis function to the j th output component

Self-organizing map

α	Adaptation gain
\mathbf{m}	Synaptic weight vector of neuron
d	Distance between \mathbf{x} and the neuron \mathbf{m}
N_c	Neighbourhood of the best-matching cell
w	Relative width of the window

Fuzzy logic

μ_i	Firing strength of the i th fuzzy rule
z_i	Output value of singleton associated to i th fuzzy rule
A, B, C	Fuzzy sets
Z	Crisp output

Neuro-fuzzy systems

$a_{j,i}$	j th reference vector of the i th input
A_i	Linguistic variable of the i th input variable

Fuzzy self-organizing map

α_i	Firing strength of the i th fuzzy rule
α_0	Firing strength of the default rule
β_1	Maximum firing strength of the default rule
$\beta_{2,i}$	Overlapping parameter of the i th normal rule
a_0	Output value of the default rule
$a_{i,j}$	Output value of the j th output of the i th fuzzy rule
$c_{i,j}$	Center of the normal fuzzy rule
g_a	Learning rate of output singletons
g_U	Learning rate of fuzzy sets
$s_{r,k}$	Spread of the first runner-up rule
$sl_{i,j}$	Left spread of the fuzzy set $U_{i,j}$
$sr_{i,j}$	Right spread of the fuzzy set $U_{i,j}$
$U_{i,j}$	Fuzzy set of the j th input of the i th fuzzy rule
y	Output of the training data set
y^*	Output of the FSOM

Chapter 1

Introduction

The amazing universe of human brain, memory, and nervous system has attracted and inspired many researchers to create computational models of neural systems. Also several Finnish researchers such as Kohonen [27], Kanerva [26] and Oja [47] have given great impacts in this area of research. Kohonen has studied neural phenomena and introduced his self-organizing map model, while Kanerva has concentrated on memory models. These models are strongly motivated by biological foundations and cognitive science.

In 1960's, Lotfi A. Zadeh proposed a logical system, fuzzy logic, which is close to the spirit of human thinking. The main idea of fuzzy logic is to describe human thinking and reasoning with a mathematical framework. Although fuzzy logic has met resistance and criticism by some appreciated researchers, fuzzy logic has found its way to several applications in many fields, such as engineering and economy.

The integration of neural networks and fuzzy logic has given birth to a new research field called neuro-fuzzy systems. These systems have potential to capture the benefits of both fascinating fields into a single framework. In 1993, Vuorimaa [61] proposed a neuro-fuzzy system, called the fuzzy self-organizing map (FSOM) which is a fuzzy version of the Kohonen's Self-Organizing Map (SOM). The key idea of the FSOM is to replace the neurons of the SOM by fuzzy rules and sets.

This master thesis work was started in autumn 1993, when so called neuro-fuzzy group was established at Signal Processing Laboratory, Tampere University of Technology. The aim of the research group is to apply the FSOM into real problems in the fields of signal processing and fuzzy control. Also the further development of the FSOM is one of the main interests.

This thesis is divided into seven chapters. Chapter 2 provides a brief introduction to neural networks. First some basic features of biological neural networks are presented. After that, two artificial neural networks and their learning methods are described in more detailed. In the chapter 3, the principles of the fuzzy logic theory and fuzzy systems are discussed.

The fourth chapter describes fusion of neural networks and fuzzy logic. First, a comprehensive review of neuro-fuzzy systems is given. After that, the neural network structured fuzzy systems are examined in more detailed. The fifth chapter is focused on the FSOM. The structure and operation principle of the FSOM is presented. The modified learning vector quantization (LVQ) competitive learning method of the FSOM is also introduced. In addition, this chapter introduces a novel version of the FSOM which is efficient for modelling locally linear systems.

In the chapter 6, the FSOM is used in three examples: identification of the laboratory water heating system, the tuning of the neuro-fuzzy controller for the a simplified inverted pendulum model, and fault diagnosis of the tank and reactor system. In the chapter 7, the properties of the FSOM are discussed, and proposals for further improvement are given. Chapter 8 summarizes this thesis.

Chapter 2

Basics of neural networks

This chapter gives a brief introduction to neural networks. First, the biological nerve nets are presented as a background to artificial neural networks. After that, two different artificial neural network models, radial basis function network and Kohonen's self-organizing map, are described with their learning algorithms.

2.1 Biological and artificial neural networks

The human brain and nervous system have amazing properties. Although brain cells operate about seven orders of magnitude slower than switching elements of a modern computer, the brain is capable of performing tasks which are impossible for computers. The main reasons for this are massive parallelism and asynchronous operation of nerve nets.

Biological neurons

The human brain consists of about hundred billions (10^{11}) neurons. Each of them has about 1000 – 10 000 connections to other neurons. The nervous system has several kind of nerve cells (neurons), but they all share common features. Figure 2.1 shows the structure of a typical biological neuron. The neuron is formed of a *soma* (*i.e.*, cell body), *dendrites*, and an *axon*. The dendrites receive signals from other neurons, and the axon passes a signal to the other neurons. A junction between the axon and the dendrite is called a *synapse*.

The operation of a neuron is very complicated, but the basic features are understandable.

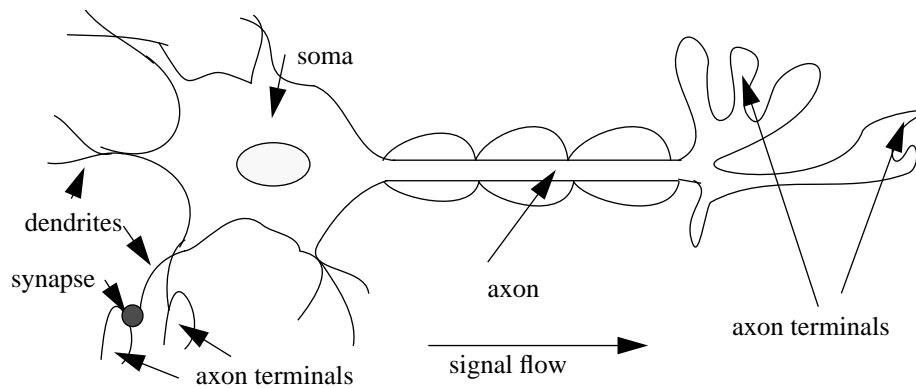


Figure 2.1 Biological neuron.

The input signals from other neurons are transmitted chemically through either excitatory or inhibitory synapses to the dendrites. Excitatory synapses raise the probability of the neuron firing (*i.e.*, firing frequency), while inhibitory synapses decrease it. The dendrites pass the signals into the soma which sums the signals both spatially and temporally (in the time domain). If the sum signal is higher than the action potential of the axon, the neuron gives a response (fires). Finally, the axon terminals transmit the response to the following synapses. After firing, the neuron must recover before it can fire again [14].

Organization of neurons

The topographical organization of the brain, especially the cerebral cortex, is quite well-known [28]. The organization has been deduced from functional deficits and behavioural impairments induced by various kind of lesion, or by haemorrhages, tumours, or malfunctions. The different regions in the brain seem to be dedicated to specific tasks. On the other hand, the neurons seem to be organized depending on the signals which they receive. Neurons which are “locally-tuned” or “selective” for some range of the input signal are found in many parts of nervous systems [41]. For example, they can have selective response to frequency (cochlear stereocilia cells in auditory system) or stimulation from localized regions of the body surface (somatosensory cortex). The population of locally-tuned cells are typically arranged in corticoidal maps in which the different input signals assign different positions in the map.

Artificial neuron

The first neuron model was proposed already in 1940's by McCulloch and Pitts. They described the behaviour of the neuron with the model

$$y = g \left(\sum_{i=0}^N x_i w_i \right) \quad (2.1)$$

where y is the output of the neuron, N is the number of the inputs and x_i are the input variables. w_i are called synaptic weights (Fig. 2.2). The function g is a nonlinear function. Commonly used functions are hardlimiter, pseudolinear, and sigmoid functions [11].

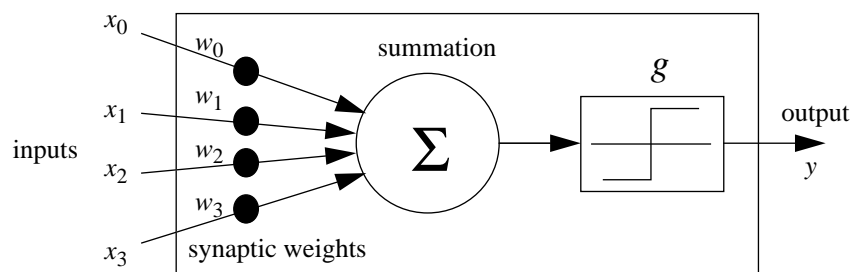


Figure 2.2 An example of an artificial neuron.

Artificial neural networks

Artificial neural networks (ANNs) [11, 40] try to mimic simple nervous systems. Kohonen [28] divides neural network architectures into three categories depending on the philosophy how they model nervous system (Fig. 2.3). *Feedforward networks* transform sets of input signals into sets of output signals. The desired input-output transformation is usually determined by external, *supervised* adjustment of the system parameters. In *feedback networks* (recurrent), the input information defines the initial activity state of a feedback system. After state transitions the asymptotic final state is identified as the outcome of the computation. In competitive, *unsupervised* or self-organizing category, neighbouring cells (neurons) in the network compete in their activities and develop iteratively specific detectors for different input signal patterns.

The neural networks are used for two main tasks: *function approximation* and *pattern classification*. In function approximation, the neural network is trained to approximate a mapping

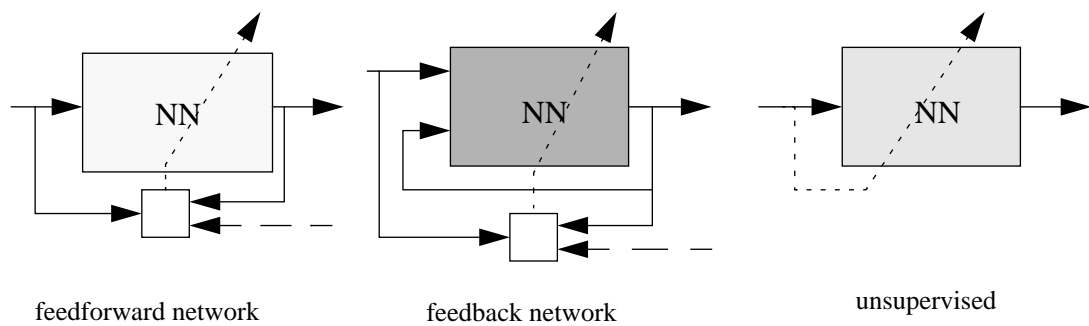


Figure 2.3 Neural network models. The longdash line is the desired output of the network and the training scheme is illustrated by dotted line.

between its inputs and outputs. Many neural network model have been proved as universal approximators, *i.e.* the network can approximate any continuous function arbitrary well [13]. The pattern classification problem can be regarded as a specific case of the function approximation. The mapping is done from the input space to a finite number of output classes.

Table 1: Categorization of some neural networks.

neural network	learning	structure	usage
Multilayer perceptron	supervised	feedforward	classification function approximation
Radial basis function network	supervised or unsupervised	feedforward	classification function approximation
Self-organizing map	unsupervised	feedforward	classification
LVQ	supervised	feedforward	classification

2.2 Radial basis function network

Radial basis function (RBF) network (sometimes called local receptive field network) provides an alternative approach opposed to the popular feedforward multilayer perceptron (MLP) network [11, 52]. As a neural network model RBF network was proposed by Moody and Darken [41]. They were inspired by the locally-tuned or selective neurons having response for some range of the input variables, which are found in several parts of human nervous system. On the other hand, the RBF network has also its origin in density function approximation and hypersurface fitting techniques as potential functions [42].

Like some other neural network architectures, the RBF network has been proved as universal

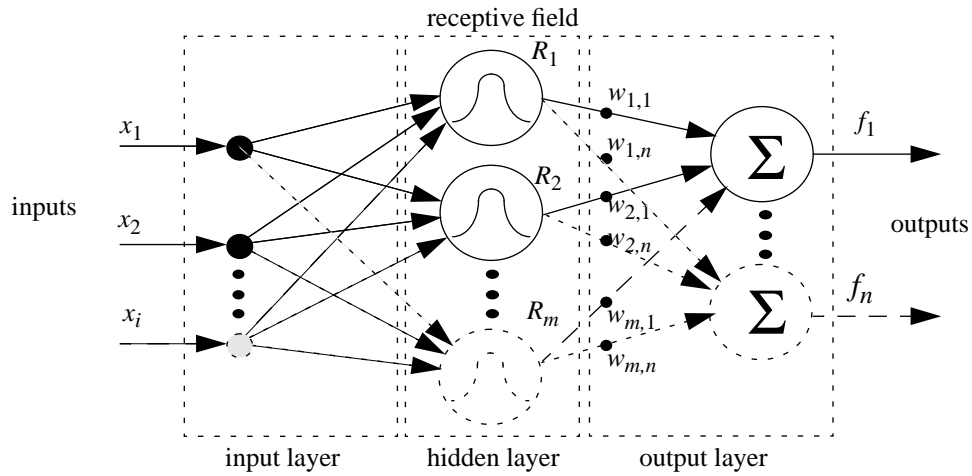


Figure 2.4 The structure of the RBF network.

approximator [36].

Structure of RBF network

The RBF network consists of three layers, as shown in Fig. 2.4. The input layer passes the inputs to the hidden layer through a set of unweighted connections. The hidden layer is composed of local basis functions (receptive fields). Usually, a Gaussian exponent function is used, which performs a nonlinear transformation:

$$R_i(\mathbf{x}) = \exp\left\{-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{\sigma_i^2}\right\}, \quad (2.2)$$

where \mathbf{x} is the input vector, \mathbf{c}_i is the center of the i th basis function and σ_i characterizes the width of the function.

The norm $\|\cdot\|$ means the Euclidean norm:

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \mathbf{x}}. \quad (2.3)$$

The Gaussian exponent function is local, because $R_i(\mathbf{x}) \rightarrow 0$, when $\|\mathbf{x} - \mathbf{c}_i\| \rightarrow \infty$. Other basis functions, such as the inverse multiquadric and thin-plate-spline function, can also be used [9].

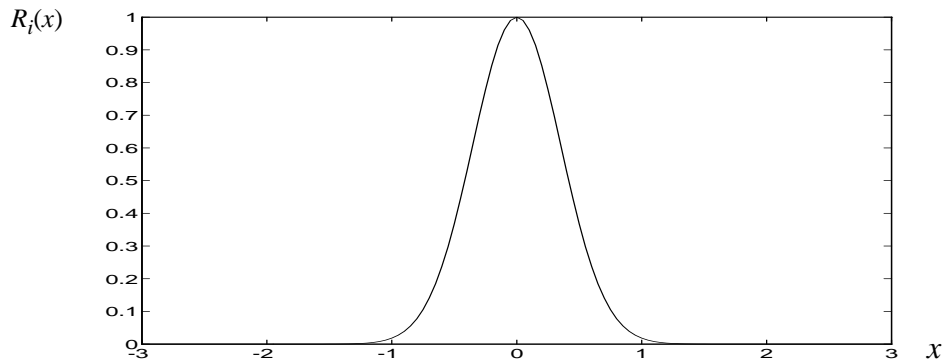


Figure 2.5 An one-dimensional Gaussian exponent function, when $c_i = 0$ and $\sigma_i = 0.5$.

The output layer combines linearly the outputs of the receptive field units. The RBF network can be expressed in two different ways. In the first case, the output of the networks is computed as a weighted sum of the outputs of the basis functions according to

$$f_j(\mathbf{x}) = \sum_{i=0}^m w_{i,j} R_i(\mathbf{x}), \quad (2.4)$$

where m is the number of the basis functions. An alternative form of the RBF network [41] gives a normalized response function

$$f_j(\mathbf{x}) = \frac{\sum_{i=0}^m w_{i,j} R_i(\mathbf{x})}{\sum_{i=0}^m R_i(\mathbf{x})}. \quad (2.5)$$

Learning methods

Several methods have been proposed to estimate the parameters of the RBF network, when it is used in function approximation. The methods can be divided into four categories:

1. The centers of the basis functions are set uniformly in the input space.
2. The centers are chosen from the training data set.
3. The centers are chosen by a clustering algorithm.
4. The centers are parameters in an optimization procedure.

In the first technique, the centers \mathbf{c}_i are allocated uniformly in the input region, so that the centers form a lattice. The weights \mathbf{w}_i are estimated after the centers are fixed. The centers are not directly used as free parameters to minimize the cost function. This method leads easily to overparametrization, especially when the dimension of the input space is high.

The centers can also be chosen directly from the samples of the training data set. That can be done totally randomly, but a better way is the use of the orthogonal least squares (OLS) [9] algorithm. The key idea of the OLS algorithm is to select the centers iteratively one by one from the training set so that the squared cost function is minimized.

In the third method, the centers of the basis functions are searched by clustering the training data vectors. Usually k -means or SOM clustering algorithms are used. The locations of the centers depend on the distribution of the data vectors in the input space. The output weights are estimated after the centers have been fixed. Moody and Darken [41] have used this approach in their hybrid scheme. The method is simple, but it does not generally produce optimal solutions, because of the allocation of the centers is based on input samples, not the minimization of the cost function.

The whole parameter set of the RBF network can be also estimated by minimizing the cost function. The gradient descent algorithm for RBF networks [36] is often used, but it easily loses the locality of basis functions [41]. Genetic algorithms can also be employed as an estimation method.

RBF as pattern classifier

The RBF network can also be used as pattern classifier. The architecture of the pattern classifier RBF differs from the previously presented. The basis functions are associated directly to the output classes. The training methods for this approach have also been developed like restricted coulomb energy method [17] and clustering method [42].

2.3 Self-organizing map

Self-organizing map (SOM) is an unsupervised neural network model developed by Kohonen [27, 28]. It creates effectively spatially organized internal representations of various features of input signals and their abstractions without any supervisor signal. The self-

organizing map can be used to visualize a high-dimensional data. The map projects areas, which are close to each other in the input space, to areas that are close to each other in the output space. Fig. 2.6 shows how a seven-dimensional input vector is projected into a discrete two-dimensional space.

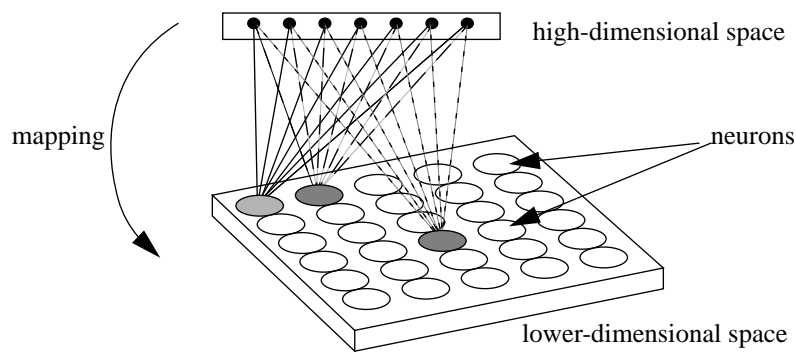


Figure 2.6 Self-organizing map. A high-dimensional vector is mapped into a lower-dimensional space.

The SOM consist of neurons which are connected to each other. The interconnections between neurons are defined by the topology. Some simple topologies of neurons are illustrated in Fig. 2.7.

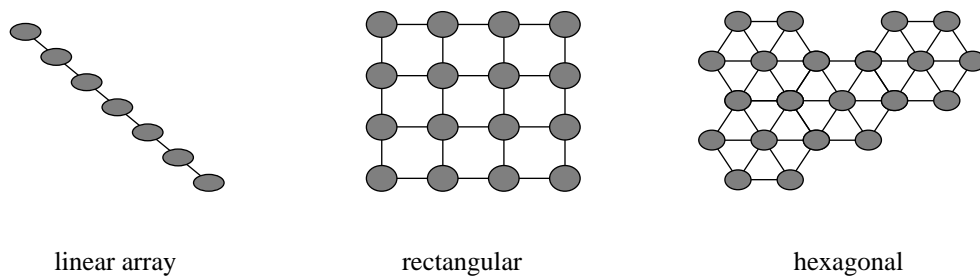


Figure 2.7 Examples of self-organizing maps.

Structure of SOM

A structure of the SOM is shown in Fig. 2.8. The self-organizing map consists of only one real layer of neurons. This structure implements *dot-product* similarity measure. In fact, it measures the cosine of the angle between normalized input and synaptic vectors. The output of each neuron can be computed according to

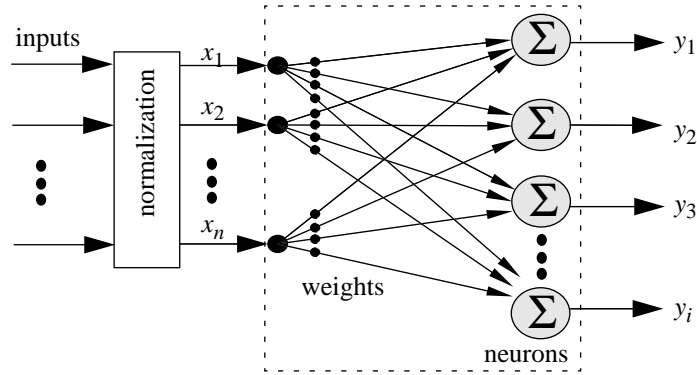


Figure 2.8 The structure of the SOM.

$$y_i = \sum_{j=1}^n m_{i,j} x_j, \quad (2.6)$$

where n is the number of inputs, $m_{i,j}$ is a synaptic weight and x_j is j th component of the input vector \mathbf{x} . The input vector \mathbf{x} has to be normalized ($\|\mathbf{x}\| = 1$) before it is fed into the network. Usually, the output of the network is given by the most active neuron.

Unsupervised learning

The SOM is trained by an unsupervised SOM algorithm [28]. The SOM algorithm can be briefly described as follows:

- Step 1. Determine the topology of the map and the number of the neurons.
- Step 2. Initialize the neurons (synaptic weights) with small random values.
- Step 3. Choose the next normalized training sample vector \mathbf{x} .
- Step 4. Choose the best-matching neuron \mathbf{m}_c , which is the closest to the input vector $\mathbf{x}(t)$ according to

$$\|\mathbf{x}(t) - \mathbf{m}_c(t)\| = \min_i \{\|\mathbf{x}(t) - \mathbf{m}_i(t)\|\}. \quad (2.7)$$

- Step 5. Update the $\mathbf{m}_c(t)$ and other neurons belonging to its topological neighbourhood $N_c(t)$ towards the sample $\mathbf{x}(t)$ according to

$$\begin{aligned} \mathbf{m}_i(t+1) &= \mathbf{m}_i(t) + \alpha(t) [\mathbf{x}(t) - \mathbf{m}_i(t)], & \forall i \in N_c(t), \\ \mathbf{m}_i(t+1) &= \mathbf{m}_i(t), & \forall i \notin N_c(t), \end{aligned} \quad (2.8)$$

where $\alpha(t)$ is a scalar valued adaptation gain $0 \leq \alpha(t) \leq 1$. This updating is illustrated in Fig. 2.9.

Step 6. Stop, if some optimal number of iteration steps is done, else continue from step 3.

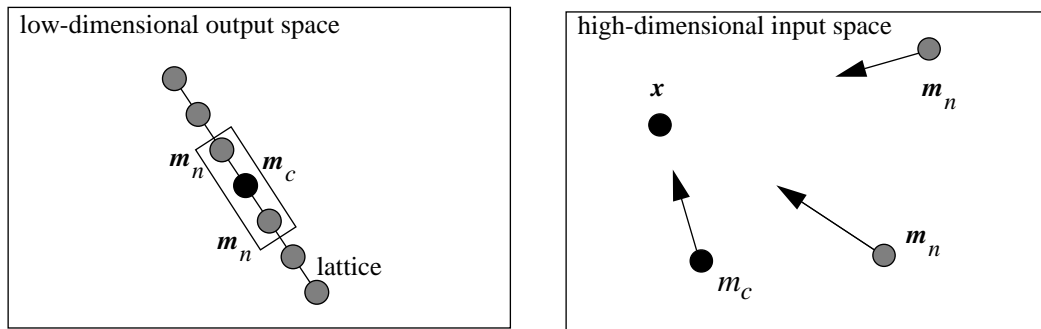


Figure 2.9 Illustration of the SOM learning algorithm. On the left, the winner and the neighbouring neurons m_n , belonging to $N_c(t)$, are found in the neuron lattice. On the right, the closest code vector and its neighbourhood are updated towards the input sample.

Usually, the adaptation gain α is decreased during time. In addition, the neighbourhood set $N_c(t)$ around the best-matching cell $m_c(t)$ should initially contain about half of the all cells and decrease with time, as shown in Fig. 2.10.

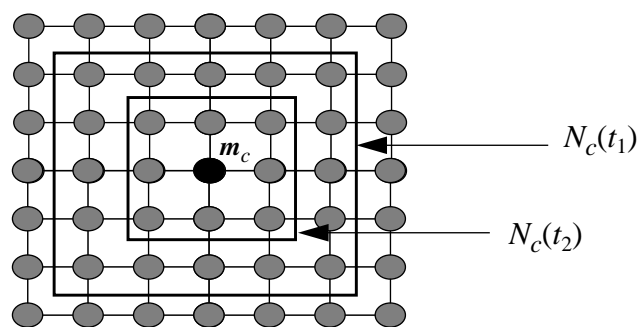


Figure 2.10 Topological neighbourhood $N_c(t)$ around the best-matching neuron $m_c(t)$ at different time steps. The neighbourhood decreases in size over time.

Learning Vector Quantization

The SOM can also be used as a pattern classifier. However, the classification accuracy of the SOM can be significantly increased by using supervised learning vector quantization (LVQ)

algorithms: LVQ1, LVQ2 or LVQ3 [28]. The goal of the algorithms is to move the decision border between neurons towards the Bayesian limit so that the number of misclassifications is minimized.

Usually, each output class or category is approximated by several neurons, as illustrated in Fig. 2.11. Thus, it is obvious that the selection of an optimal number of neurons per class is important. The SOM can be used to decide how many of the fixed number of cells are dedicated to each class [25]. Another approach is to add new neurons to the map dynamically during training [51].

In this thesis, we are especially interested in LVQ2 and LVQ3 algorithms which are based on a concept of *window*, as illustrated in Fig. 2.12. The window defines a region in the input space between two neurons, where the decision border is updated. The window is defined as follows

$$\min \left\{ \frac{d_i}{d_j}, \frac{d_j}{d_i} \right\} > \frac{1-w}{1+w}, \quad (2.9)$$

where d_i and d_j are distances between the input sample \mathbf{x} and code vectors \mathbf{m}_i and \mathbf{m}_j , respectively. Parameter w is the relative width of the window. If the input sample \mathbf{x} falls into the window, the synaptic weights of the neurons are updated according to

$$\begin{aligned} \mathbf{m}_i(t+1) &= \mathbf{m}_i(t) - \alpha(t) [\mathbf{x}(t) - \mathbf{m}_i(t)], \\ \mathbf{m}_j(t+1) &= \mathbf{m}_j(t) + \alpha(t) [\mathbf{x}(t) - \mathbf{m}_j(t)], \end{aligned} \quad (2.10)$$

where \mathbf{x} and \mathbf{m}_j belong to the same output class, but \mathbf{x} and \mathbf{m}_i into different classes. The operation of learning law (2.10) is illustrated in Fig. 2.12.

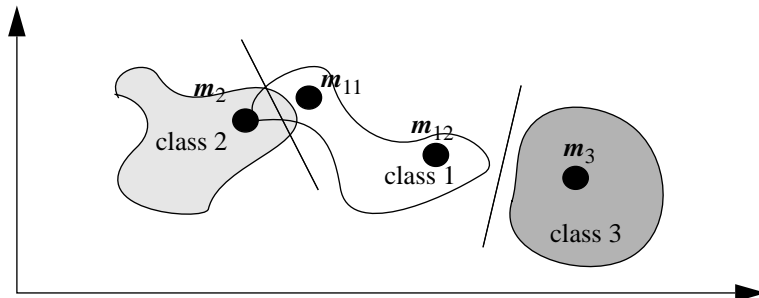


Figure 2.11 Codebook vectors and pattern classes. The decision boundaries are illustrated by lines.

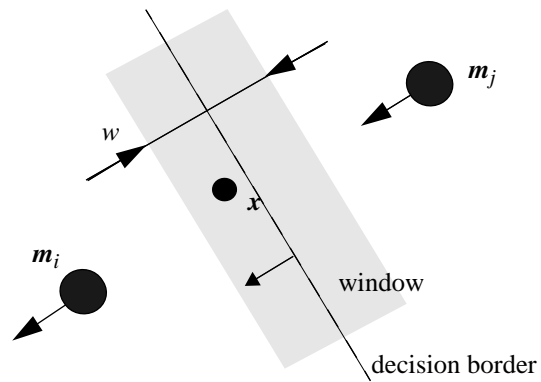


Figure 2.12 Illustration of the LVQ2 learning. The input sample x belongs to the same output class as m_j . When the sample falls to the window, the code vector m_j is updated towards the sample, and m_i is pushed away. At the same time, the decision border moves.

Chapter 3

Basics of fuzzy logic

Fuzzy logic is a logical system providing a mathematical framework to capture the uncertainties associated with human cognitive systems such as thinking and reasoning. Simply, it simulates human thinking which operates more likely on symbols than exact values. In fact, our daily thoughts and communication are full of these symbols or fuzzy expressions. This chapter gives a brief introduction to the main concepts of fuzzy logic.

3.1 Fuzzy sets

In conventional set theory, an element either belongs to the set or not. Fuzzy logic is a generalization of the conventional logic. In fuzzy set theory, the element can belong to the set partially with a certain degree. The difference between conventional crisp and fuzzy sets is illustrated in Fig. 3.1. Let us consider tree example sets *the poors*, *the averages* and *the richs* in the universe of discourse ‘wealth’. In conventional logic, persons are divided into these three groups crisply. The fuzzy sets have no crisp boundaries, but a person can simultaneously be a member of several groups with different degrees. For example, a person can be rich with degree of 0.1 and average with 0.7.

Linguistic variables

The main advantage of fuzzy logic is that words or sentences can be used as expressions instead of numeric values. The associative expressions are called linguistic variables. They

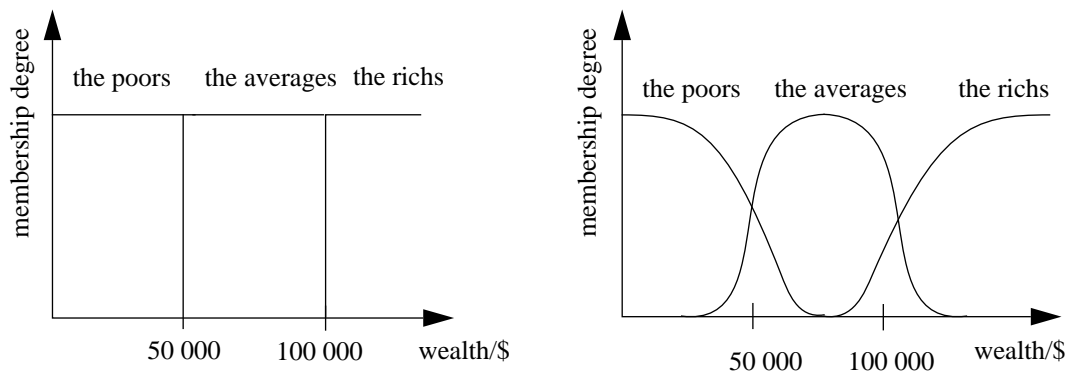


Figure 3.1 Conventional sets and fuzzy sets.

are common in our daily life. Let us consider the fuzzy variable *velocity*. It can be, for example, divided into three linguistic variables: slow, medium, and fast which are fuzzy sets, as show in Fig. 3.2. Each linguistic variable is represented by membership function in the universe of discourse. For example, the membership function of the linguistic variable slow could be defined by

$$\mu_{slow}(v) = \begin{cases} 1, & v \in [0, 35], \\ 1 - \left(\frac{v - 35}{25}\right), & v \in [35, 60]. \end{cases}$$

The membership function values can vary between zero and unity and they can have many shapes, as shown in Fig. 3.3. The selection of the shape for a fuzzy set is subjective and particular rules do not exist, but the singleton type of membership function (fuzzy unit set) is usually employed only for the output variables of the fuzzy reasoning.

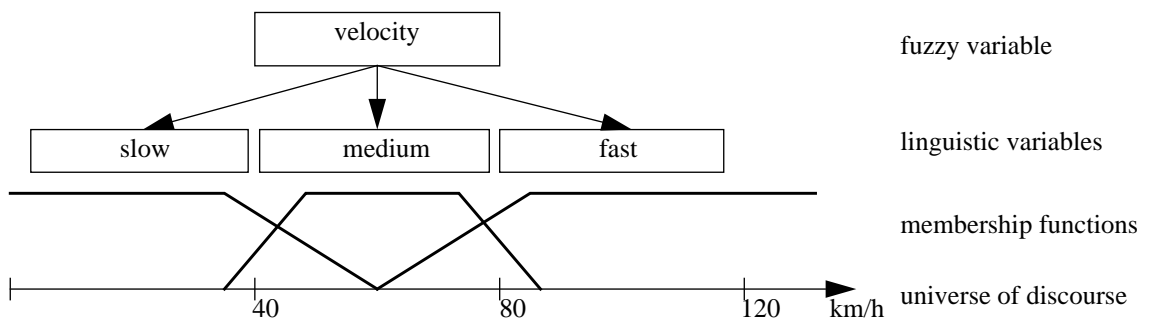


Figure 3.2 Terms of fuzzy logic.

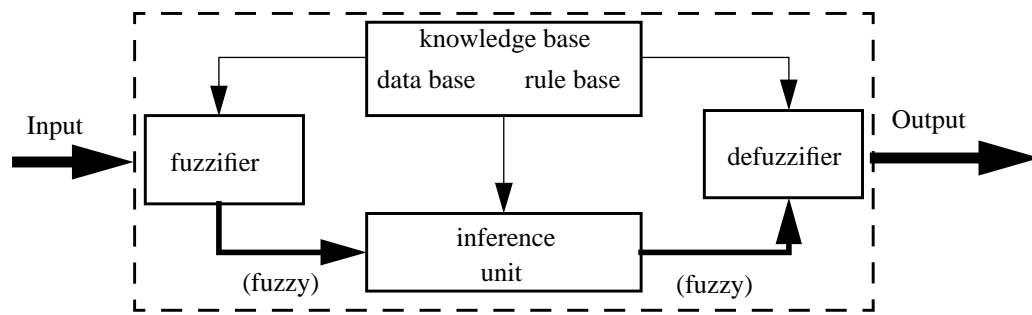


Figure 3.4 The structure of the fuzzy logic inference system.

In the second stage, they are fuzzified. After that, the fuzzified inputs are combined according to the fuzzy rules in the knowledge base. Finally, the results of all rules are combined and defuzzified. In the following, each stage is described in more detail.

Fuzzification

In the fuzzification, the crisp input values are transformed to fuzzy values. If the input has a crisp value, the matching against the membership function of linguistic variable is shown in Fig. 3.5(a). If the input contains noise, it can be modelled by using a fuzzy input value. In this case the fuzzy output is the intersection of fuzzy input and the linguistic variable membership functions as shown in Fig. 3.5(b). However, the crisp input value fuzzification is mostly used because of its simplicity.

Inference

The decision making unit performs the inference operations on the fuzzy rules. The fuzzy values within a fuzzy rule are aggregated with connective operators like intersection (AND), union (OR) and complement (NOT). Due to the use of the multi-valued logic, the connective operators for fuzzy logic differ from the ones used in the Boolean logic. The operators can be

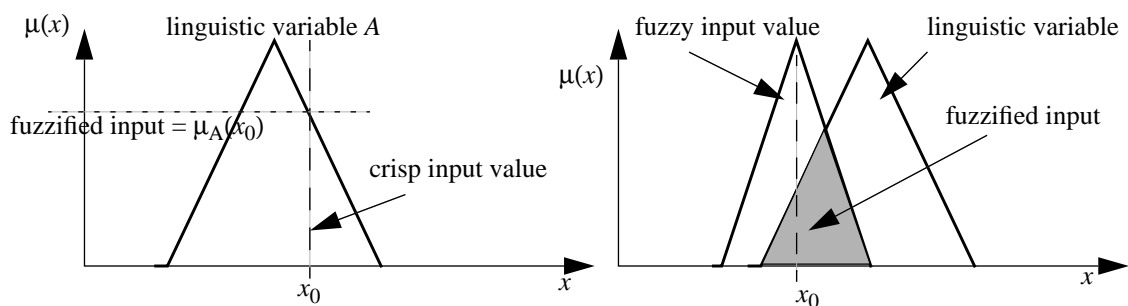


Figure 3.5 Fuzzification of a crisp input (on the left) and a fuzzy input (on the right).

defined in several ways, but the following are the best-established ones [34]:

Intersection

$$\mathbf{AND}(\mu_A, \mu_B) = \min\{\mu_A, \mu_B\}, \tag{3.1}$$

$$\mathbf{AND}(\mu_A, \mu_B) = \mu_A \mu_B, \tag{3.2}$$

Union

$$\mathbf{OR}(\mu_A, \mu_B) = \max\{\mu_A, \mu_B\}, \tag{3.3}$$

Complement

$$\mathbf{NOT}(\mu_A) = 1 - \mu_A, \tag{3.4}$$

where μ_A and μ_B are membership values which are combined by operators. The firing strengths of the fuzzy rules are computed by employing above operators. The operation of the intersection is shown in Fig. 3.6. The final output fuzzy sets are obtained either scaling (Max-Dot method) or cutting (Max-Min) according to the firing strength of the fuzzy rules. If the output fuzzy sets are singletons, they are not handled by the firing strengths in this stage.

Defuzzification

In the defuzzification stage, the outputs of the fuzzy rules are combined to a crisp output

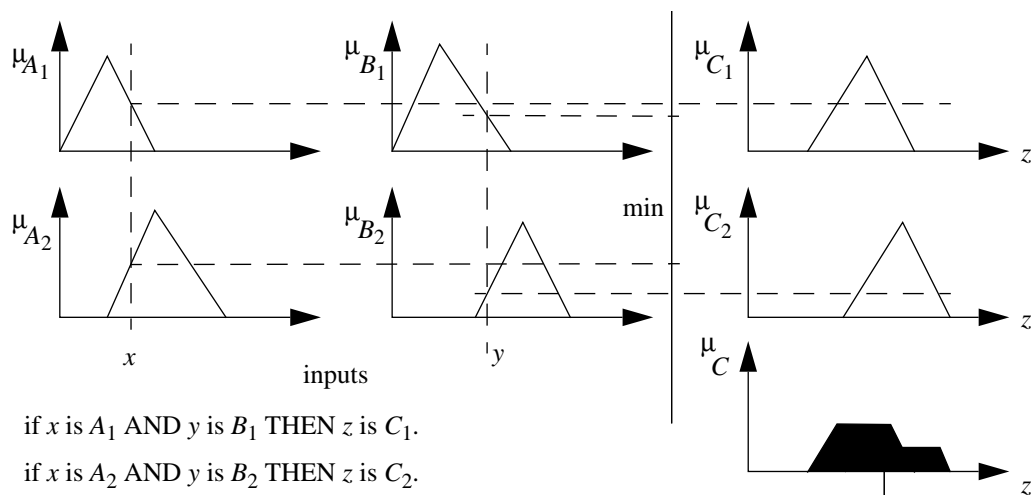


Figure 3.6 The fuzzy inference using the Min-inference.

value. Several defuzzification strategies has been suggested [35]. The most common method is the center of area (COA) defuzzification strategy, illustrated in Fig. 3.7. Assuming a discrete universe of discount, the crisp output Z is produced by searching the center of gravity of consequence fuzzy sets according to

$$Z = \frac{\sum_{i=0}^m \mu_C(z_i) z_i}{\sum_{i=0}^m \mu_i(z_i)}, \quad (3.5)$$

where m is the number of quantization levels of the output, z_i is the amount of output at the quantization level i , and $\mu_i(z_i)$ represents its membership value in C .

If only singletons are used as the consequences of fuzzy rules, the natural defuzzification method is the weighted average (WA). It can be considered as a special case of COA defuzzification method. The WA method combines the consequences of the fuzzy rules to the output of the inference system z according to

$$Z = \frac{\sum_{i=0}^n \mu_i z_i}{\sum_{i=0}^n \mu_i} \quad (3.6)$$

where n is number of fuzzy rules, μ_i is the firing strength of the rule, and z_i is the output value of the i th singleton.

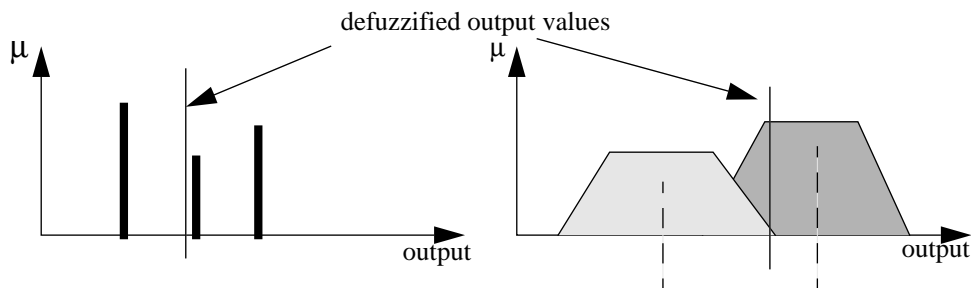


Figure 3.7 Defuzzification methods. On the left, three fuzzy rules which have singleton output fire. The output is computed by using weighted average strategy. On the right, two fuzzy rules fires. The crisp output is the center of the area.

Chapter 4

Neuro-fuzzy systems

This chapter introduces systems, which combine the properties of neural networks and fuzzy logic, called neuro-fuzzy systems. First, the background and the taxonomy of neuro-fuzzy systems is presented. After that, the interest is focused on neural fuzzy logic inference systems which are capable of extracting fuzzy rules and sets from numerical input-output information.

4.1 Background of neuro-fuzzy systems

In previous chapter, the introduction to neural networks and fuzzy logic was given. It was noted that neural networks have two main benefits. First, they are capable of learning non-linear mappings of numerical data. Second, they perform parallel computation. However, the operation of neural networks have also many weaknesses. For example, in the popular multilayer perceptron network, the knowledge of the system is distributed into the whole network as synaptic weights. Therefore, it is very hard to understand the meaning of weights, and the incorporation of prior knowledge into the system is usually impossible. Although the knowledge of RBF and SOM neural networks is in a more comfortable form, it cannot be easily extracted into linguistic rules. Fuzzy logic uses human understandable linguistic terms to express the knowledge of the system. This makes possible a close interaction between the system and human operator which is very desirable property.

The aim of neuro-fuzzy systems (Fig. 3.8) is to combine collectively the benefits of both approaches. Simply, the operation of the system is expressed as linguistic fuzzy expressions

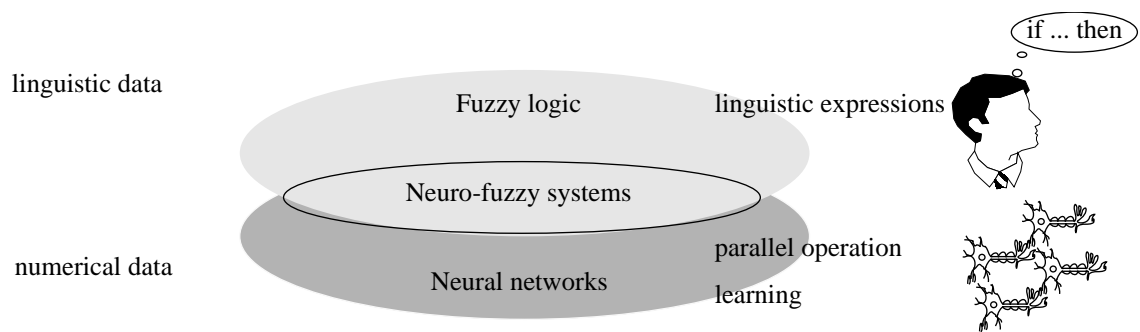


Figure 3.8 Neuro-fuzzy systems integrate fuzzy logic and neural networks.

and learning schemes of neural networks are used to learn the system. In addition, neuro-fuzzy systems allow incorporation of both numerical and linguistic data into the system. The neuro-fuzzy system is also capable of extracting fuzzy knowledge from numerical data.

The neuro-fuzzy systems can be divided into two main groups:

1. Neural fuzzy inference systems and
2. Fuzzy neural networks.

The origin of neural fuzzy inference systems is to incorporate neural concepts, such as learning and parallelism, into fuzzy logic inference systems (fuzzy controllers, in the context of control applications). Neural fuzzy inference systems realize fuzzy inference [2, 16, 22]. The architecture of the systems are parallel, and they exploit the same learning algorithms, which are used with neural networks. The fuzzy inference can be implemented in two ways. The most common approach is to use one network which realize the whole fuzzy inference. In the second approach, each fuzzy rule is realized using a neural network, when the fuzzy inference is the result of several neural networks [56]. The former subgroup is studied in more detail in this thesis.

In the fuzzy neural networks [15, 49, 50], the fuzzy ideas are incorporated into neural networks. Lee and Lee [37] are the first who worked on neuro-fuzzy systems, especially on fuzzy neural networks. Their approach replaced the weighted sum of the McCulloch-Pitts neuron by an corresponding fuzzy operation. The operation of their neuro-fuzzy system was exactly the same as the McCulloch-Pitts neural network. Unfortunately, they did not give any training laws for the network. The fuzzy neural networks consist of two components in same system: a fuzzy system and a neural network. The fuzzy system can be either a fuzzy inference block which converts linguistic information for the neural network or the neural net-

work can drive the fuzzy inference block.

A common approach to neuro-fuzzy systems is to fuzzify the learning algorithms of different neural networks paradigms. For example, Huntsberger and Ajjimarangsee [20] have proposed that the learning rate coefficient of Kohonen's self-organizing map is treated as a fuzzy membership value of the current input sample in the class of each neuron. The membership values are computed with the fuzzy *c*-means algorithm. Tsao *et al.* [60] have extended their ideas to a family of Kohonen's SOM algorithms. Similarly, Bedzek *et al.* [5] and Chung and Lee [12] have extended this approach to Kohonen's learning vector quantization algorithm. Similarly, Carpenter *et al.* [7, 8] have also developed fuzzy versions of their ART and ART-MAP models.

4.2 Neural fuzzy inference systems

The basic idea in neural fuzzy inference systems is to incorporate parallel architecture and learning capability to fuzzy inference systems. The early systems implemented as multilayer networks [22], and they were learned using gradient descent method. Later, close functional similarities between fuzzy inference systems and RBF networks were also noticed [23, 46, 67]. In this section, the both structures and learning schemes of neural fuzzy inference systems are presented. Because of the systems can be used as function approximator or classifiers, the both architectures are discussed.

Knowledge structure and linguistic variables

Fuzzy rules are the fundamental part of the knowledge base in a fuzzy inference system. A number of rule variations have been introduced depending on the use of the system and its derivation. Next, some typical forms of fuzzy rules used in the neuro-fuzzy systems are presented.

Fuzzy rules of neuro-fuzzy systems

The fuzzy reasoning rules can be divided into four main types. For simplicity, only the two-input single-output model of the neuro-fuzzy system is now presented. Four types of the fuzzy rules can be described as follows

$$\mathbf{IF } x_1 \text{ is } A_1 \mathbf{ AND } x_2 \text{ is } A_2 \mathbf{ THEN } y \text{ is } B, \quad (4.1)$$

$$\mathbf{IF } x_1 \text{ is } A_1 \mathbf{ AND } x_2 \text{ is } A_2 \mathbf{ THEN } y \text{ is } z, \quad (4.2)$$

$$\mathbf{IF } x_1 \text{ is } A_1 \mathbf{ AND } x_2 \text{ is } A_2 \mathbf{ THEN } y \text{ is } f(x_1, x_2), \quad (4.3)$$

$$\mathbf{IF } x_1 \text{ is } a_{j,1} \mathbf{ AND } x_2 \text{ is } a_{j,2} \mathbf{ THEN } y \text{ is } z, \quad (4.4)$$

where x_i is the i th input variable, and A_i is the one of the linguistic variables defined for it. The fuzzy output variable y is defined separately for each rule. In the first rule, the consequence of the rule is fuzzy set B , while the second rule uses a singleton. The consequence of third rule is a function of the input variables [57]. The antecedent part of the fourth rule uses the reference values $a_{j,i}$, when the firing strength of the rule is computed by measuring the distance between the inputs and the references [33]. When the neuro-fuzzy system is used as a classifier, the fuzzy rules have the if-then form [17, 32]

$$\mathbf{IF } x_1 \text{ is } A_1 \mathbf{ AND } x_2 \text{ is } A_2 \mathbf{ THEN CLASS is } n, \quad (4.5)$$

$$\mathbf{IF } x_1 \text{ is } a_{j,1} \mathbf{ AND } x_2 \text{ is } a_{j,2} \mathbf{ THEN CLASS is } n. \quad (4.6)$$

Traditionally, the input variables are divided into several linguistic variables which are common for many fuzzy rules. The input space partitionings (Fig. 3.9) shows, how the fuzzy rules are related to the input space. Each partitioning depends on used fuzzy rules and fuzzy

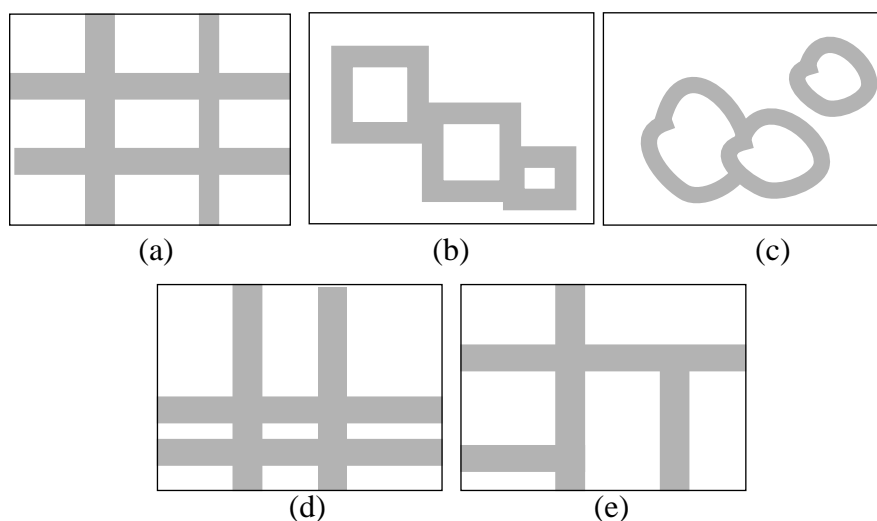


Figure 3.9 Input space partitionings. (a) fuzzy grid, (b) fuzzy boxes, (c) fuzzy clusters, (d) adaptive fuzzy grid, (e) fuzzy k-d tree.

membership functions. The fuzzy grid is fixed partitioning, the fuzzy sets can not be tuned at all. The adaptive fuzzy grid is similar, but membership functions are tunable. The fuzzy boxes and clusters are motivated by neural networks which implement a similarity measure. They are interesting, but they produce complex rule bases. The fuzzy k-d tree partitioning is flexible

Network architecture

The fuzzy logic inference system can be implemented as a five-layer neural network (Fig. 3.10). This type of architecture is the most common among neural fuzzy inference systems, and it uses the fuzzy rules defined in (4.2). Let us consider a system which has two inputs x_1 and x_2 and only one output y . In addition, the rule base contains only two fuzzy rules:

Rule 1: **IF** x_1 is A_1 **AND** x_2 is B_1 , **THEN** y is z_1 ,

Rule 2: **IF** x_1 is A_2 **AND** x_2 is B_2 , **THEN** y is z_2 .

The operation of the this system can be described layer by layer as follows:

Layer 1. Fuzzification

This layer consists of linguistic variables. The crisp inputs x_1 and x_2 are fuzzified by using membership functions of the linguistic variables A_i and B_i . Usually, triangular, trapezoid, or bell-shaped membership functions are used. For example, the bell-shaped membership functions is defined as follows

$$\mu_C(x) = \exp\left\{-\frac{\|x - c_C\|^2}{\sigma_C^2}\right\}, \quad (4.7)$$

where C is a fuzzy set, c_C is its center and σ_C is its width.

Layer 2. Rule nodes

The second layer contains one node per each fuzzy if-then rule. Each rule node performs connective operation between rule antecedents (if-part). Usually, the minimum or the dot product is used as intersection **AND**. The union **OR** is usually done using maximum operation. In our example case the firing strengths α_i of the fuzzy rules are computed according to

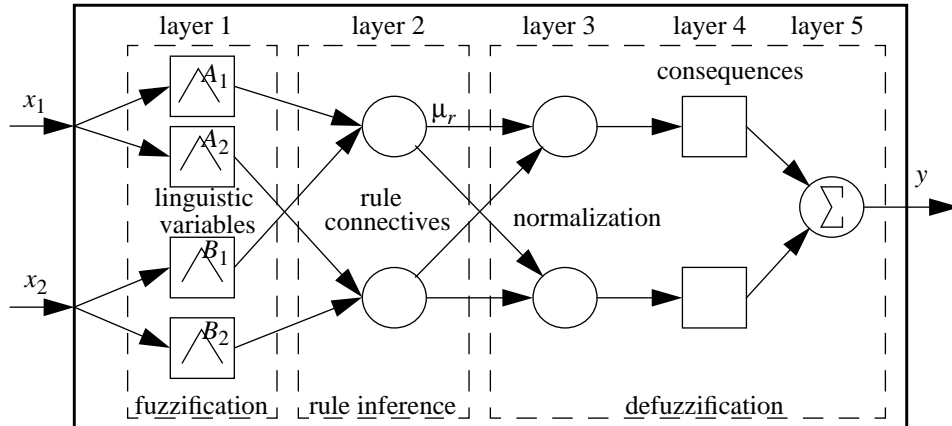


Figure 3.10 A neural fuzzy inference system.

$$\alpha_i = \min \{ \mu_{A_i}(x_1), \mu_{B_i}(x_2) \} . \quad (4.8)$$

Layer 3. Normalization

In this layer, the firing strengths of the fuzzy rules are normalized according to

$$\alpha^* = \frac{\alpha_r}{\sum_{i=0}^m \alpha_i} . \quad (4.9)$$

Layer 4. Consequence layer

This layer is related to consequent fuzzy labels z_i , which are singletons in our example case. The values of the singletons are multiplied by normalized firing strengths according to

$$y_r = \sum_{i=0}^m z_i \alpha^* . \quad (4.10)$$

Layer 5. Summation

This layer computes the overall output as the summation of the incoming signals:

$$y = \sum_{i=0}^m y_r. \quad (4.11)$$

Neuro-fuzzy classifier architecture

The architecture of the neuro-fuzzy classifier is slightly different from the architecture used in function approximators. Two first layers have the identical function with the approximator. Fig. 3.11 shows a neuro-fuzzy system using following fuzzy rules,

Rule 1: **IF** x_1 is A_1 **AND** x_2 is B_1 , **THEN CLASS** is 1,

Rule 2: **IF** x_2 is A_2 **AND** x_2 is B_2 , **THEN CLASS** is 2,

Rule 3: **IF** x_1 is A_1 **AND** x_2 is B_2 , **THEN CLASS** is 1.

Layer 3. Combination of firing strengths

If several fuzzy rules have the same consequence class, this layer combines their firing strengths. Usually, the maximum connective (**OR** operation) is used.

Layer 4. Fuzzy outputs

In this layer, the fuzzy values of the classes are available. The values describes, how well the input of the system matches to the classes.

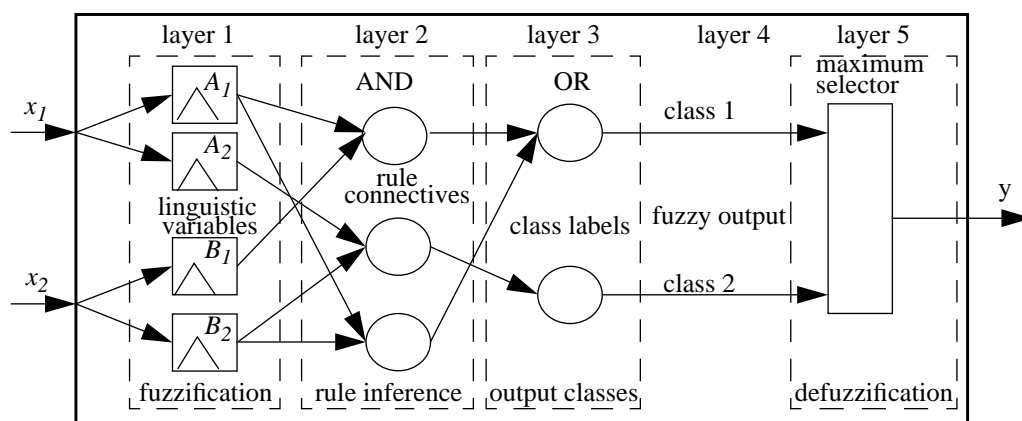


Figure 3.11 A neural architecture of the neural fuzzy classifier.

Layer 5. Defuzzification

If the crisp classification is needed, the best-matching class for the input is chosen as the output class.

Table 2: Basic properties of the neural fuzzy inference systems.

Neuro-fuzzy system	Antecedent membership functions	Partitioning	Rule connectives	Consequent membership functions	Defuzzification
Nie [46]	bell-shaped	cluster	AND/product	singleton	WA
Berenji [4]	triangular	adaptive grid	AND/soft-min	triangular	LMOM
Wang [67]	bell-shaped	radial	AND/product	singleton	WA
Lin [38]	bell-shaped	adaptive grid	AND/min OR/max	bell-shaped	COA, WA
Jang [22]	bell-shaped	adaptive grid	AND/product	singleton, functional	WA
Vuorimaa [48, 61]	triangular	boxes	AND/ min or product	singleton, functional	WA
Nauck [43, 44]	Tsukamoto's monotonic	adaptive grid	AND/min	Tsukamoto's monotonic	Tsukamoto
Horikawa [18]	bell-shaped	adaptive grid	AND/product	singleton, functional or monotonic	WA, Tsukamoto
Kwon [33]	bell-shaped	fuzzy clusters	KNN measure	singleton	WA
NeuFuz4 [59]	trapezoidal/ bell-shaped	adaptive grid	AND/product	singleton	WA
Kosko [29]	triangular	grid	AND/min	triangular	COA

Learning algorithms

The learning of the neuro-fuzzy system can be divided into two main parts: structure identification and parameter identification (estimation). The former is related to finding a suitable number of fuzzy rules and a proper partitioning of input and output space. The latter deals

with adjustment of system parameters, such as membership functions and other possible parameters [55].

Backpropagation

Backpropagation (BP) is probably the most popular neural learning method. It is an application of gradient descent algorithm originally for multilayer perceptron network. On research of neuro-fuzzy systems, the gradient descent algorithm is used by several authors [4, 16, 18], and it is discussed widely in neural network literature [11]. Usually, the initial fuzzy sets and rules are first given by user. After that, the fuzzy rules are updated by a gradient descent algorithm. The slow convergence speed near the minima is the biggest drawback of the backpropagation. In addition, the algorithm can be trapped into a local minima.

Adaptive Vector Quantization

The method of Kosko and Kong [29, 30, 31] divides input and output spaces into overlapping fuzzy sets using a clustering scheme. After that, the fuzzy rules are found out, through the best relations between input and output fuzzy sets. This method has the drawback that the fuzzy sets are not tuned at all. Thus the performance is limited by the initial partitioning of input and output spaces.

Hybrid learning algorithms

Nie and Linkens [46] integrate fuzzy inference systems with radial basis function networks. They propose a two-phase learning algorithm (so called fuzzified dynamical self-organizing scheme). In the first phase of algorithm, self-organized map (SOM) learning algorithm is used to determine the centers of the radial basis function units in the input space. Each radial basis function has identical width. New units are generated by using simple heuristics. After that, the RBF units are associated to fuzzy rules and fuzzy sets. In the second phase, a gradient descent is employed to optimally adjust the values of the output singletons.

Lin and Lee [38] have also proposed a hybrid learning method which uses Kohonen's SOM algorithm, but employs it in a different way. First, the centers of membership functions are founded by dividing each input and output linguistic variable into predetermined number of clusters. The widths of fuzzy sets are determined by N -nearest neighbours heuristics. After that, the fuzzy rules are founded by feeding training data set through the system and deciding which fuzzy rules are the most important. In addition, some fuzzy rules can be combined if

they have similar consequences or premises. Finally, the centers and the widths of the fuzzy sets are tuned using a gradient descent algorithm.

Kwon and Zervakis [33] use a modified K -means algorithm to find the centers of input membership functions. This K -means clustering method ensures that the distance map of the input space is almost linearly related to the output distance map. The membership functions differ from typical ones having only a center parameter. The output singletons and the centers are tuned by a gradient descent algorithm.

Jang [22] uses a combination of the least-squares method and gradient descent for tuning his ANFIS architecture. The system is initialized with a number of membership functions and a fixed rule base. The learning scheme consists of two separate passes. In the forward pass, the consequent parameters are identified by least-square method. The antecedent parameters are updated by a gradient descent algorithm in the backward pass.

Orthogonal least squares method

Wang and Mendel [67] employ orthogonal least squares (OLS) algorithm in their systems. The OLS is widely used for constructing the RBF networks [9], but it can also be employed to allocate so called fuzzy basis functions. The neuro-fuzzy system using fuzzy basis functions shares the same functionality with the RBF system, where the firing strengths are normalized. The initial fuzzy system is constructed based on as many fuzzy basis functions as given input-output pairs. After that, the most significant basis functions are selected to the final system using the OLS.

Summary

Table 3 collects learning schemes used in several recently introduced neural fuzzy inference systems. Their training methods differ very much from each other and no comparison of methods has been presented. Thus, it is very difficult to compare their properties, such as speed of learning or approximation capability. The self-construction heuristic means that the system has some kind of rules for adding of new fuzzy sets or fuzzy rules during learning.

Learning schemes for classifiers

Kuo *et al.* [32] have proposed a fuzzy neural network model, which maps the weighted Euclidean distance fuzzy classification procedure into a four-layer neural network structure.

The neurons are associated to the output classes. A near hierarchical clustering method is used to learn neurons in the same output class, like in LVQ algorithm.

The learning of the neuro-fuzzy classifier, proposed by Halgamuge *et al.* [17], is based on the a modified restricted coulomb energy learning scheme. The algorithm trains so called cubic basis functions, which are identical to fuzzy boxes partitioning and RBF networks. The system produces simple fuzzy reasoning from the basis functions by first removing superfluous and redundant input variables and then segmenting the input space.

Table 3: Learning of neuro-fuzzy function approximators.

Neuro-fuzzy system	Premise learning	Consequent learning	Self-constructing
Nie [46]	modified SOM	gradient descent	yes
Berenji [4]		gradient descent	no
Wang [67]		OLS	OLS
Lin [38]	SOM, initial learning	gradient descent	yes
Jang [22]	gradient descent	Least-squares method	no
Vuorimaa [48, 63]	modified LVBQ2.1 and SOM	modified LVQ2.1	yes [63]
Nauck [43, 44]		fuzzy gradient descent	no
Horikawa [18]		gradient descent	no
Kwon [33]	modified k -means and gradient descent	LMS	yes
NeuFuz4 [59]		gradient descent	no
Kosko [29]	AVQ	AVQ,	no

Chapter 5

Fuzzy self-organizing map

This chapter introduces a neuro-fuzzy system called the fuzzy self-organizing map (FSOM). The FSOM is based on the well-known Kohonen's self-organizing map, but the neurons of original model are replaced by fuzzy rules and their fuzzy sets. The learning of the neuro-fuzzy system is based on a modified LVQ algorithm.

5.1 Basic structure

A structure of Kohonen's self-organizing map (Fig. 2.8) contains one real layer of linear neurons which implement distance computation between input and reference codebook vectors. When the inner product matching measure is used, input signals are fed to all neurons, where they are multiplied by the connection weights of each neuron. The output of the neuron is calculated according to (2.6).

The basic idea of the FSOM [61] is to replace the weighted sum of the neuron by a fuzzy rule which has the general *if-then* form [64]:

$$\begin{aligned} &\mathbf{if } x_1 \text{ is } U_{i,1} \mathbf{ and } x_2 \text{ is } U_{i,2} \dots \mathbf{ and } x_n \text{ is } U_{i,n} \\ &\mathbf{then } y_1 \text{ is } a_{i,1} \mathbf{ and } y_2 \text{ is } a_{i,2} \mathbf{ and } \dots \mathbf{ and } y_p \text{ is } a_{i,p}, \end{aligned} \quad (5.1)$$

where each condition $(x_j \text{ is } U_{i,j})$ is interpreted as the membership value $\mu_{U_{i,j}}(x_j)$ of the input signal x_j in the fuzzy set $U_{i,j}$. The consequence $a_{i,j}$ of the fuzzy rule is a real-valued singleton.

Membership functions

The original version of the FSOM uses triangular membership functions. The triangular shape has the advantage that it is computationally easy. The triangular membership function (Fig. 5.1) is defined as

$$\begin{aligned}
 \mu_{U_{i,j}}(x_j) &= \frac{x_j - sl_{i,j}}{c_{i,j} - sl_{i,j}}, & sl_{i,j} \leq x_j \leq c_{i,j}, \\
 \mu_{U_{i,j}}(x_j) &= \frac{c_{i,j} - sr_{i,j}}{c_{i,j} - sr_{i,j}}, & c_{i,j} < x_j \leq sr_{i,j}, \\
 \mu_{U_{i,j}}(x_j) &= 0, & \text{otherwise,}
 \end{aligned} \tag{5.2}$$

where $c_{i,j}$ is the center of the fuzzy set $U_{i,j}$. Parameters $sl_{i,j}$ and $sr_{i,j}$ are the left and right spreads of the fuzzy set $U_{i,j}$, respectively. The fuzzy sets define a region of the input space, where each fuzzy rules fires.

Inference

The firing strength α_i of each fuzzy rule is computed by combining the membership values $\mu_{U_{i,j}}$ together using union connective operation. In the FSOM, the *minimum* operation is used as the connective. Therefore, the firing strengths are computed according to

$$\alpha_i = \min \{ \mu_{U_{i,1}}(x_1), \mu_{U_{i,2}}(x_2), \dots, \mu_{U_{i,n}}(x_n) \}. \tag{5.3}$$

Default rule

In addition to normal triangular membership functions, the FSOM uses a *default rule* [62]. It has neither spreads nor a center, but it covers the input space where the normal fuzzy rules do not fire. The default rule is defined according to

$$\begin{aligned}
 \alpha_0 &= \max \{ \beta_1 [1 - \alpha_k / \beta_{2,k}], 0 \}, \\
 \alpha_k &= \max \{ \alpha_i \},
 \end{aligned} \tag{5.4}$$

where β_1 is the maximum firing strength of the default rule and $\beta_{2,k}$ characterizes how

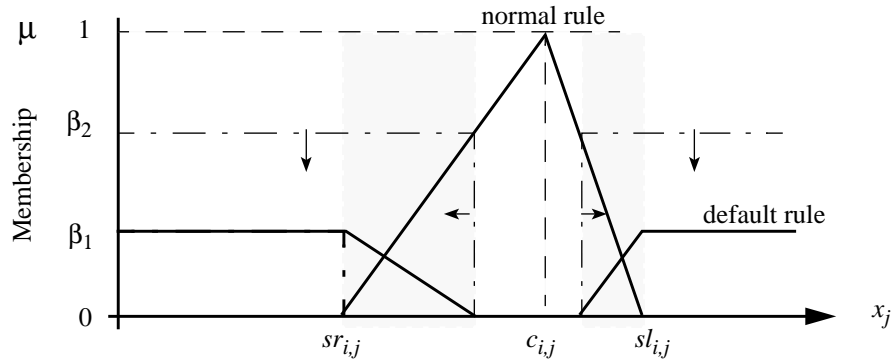


Figure 5.1 Normal fuzzy rule and default rule. The arrows illustrates how β_2 is changed during learning.

much the default rule overlaps with the normal rule. The values of $\beta_{2,k}$ decreases during learning procedure towards β_1 . The idea of the default rule is to cover the areas in the input space which are not important, or are not defined. The *completeness* of input-output mapping can be reached using default rule. That is, the system is capable of inferring an output for every input sample.

Defuzzification

The consequences of the fuzzy rules are singletons, and the output of the neuro-fuzzy system is calculated using a weighted average (WA) defuzzification method defined according to

$$y_k^* = \frac{\sum_{i=0}^m \alpha_i a_{i,k}}{\sum_{i=0}^m \alpha_i}, \quad (5.5)$$

where m is the number of the fuzzy rules. The values of the singleton $a_{i,k}$ are specified separately for each output and fuzzy rule.

The basic structure of the FSOM, which is based on the fuzzy inference described above, is illustrated in Fig. 5.10, and the operation in Fig. 5.3.

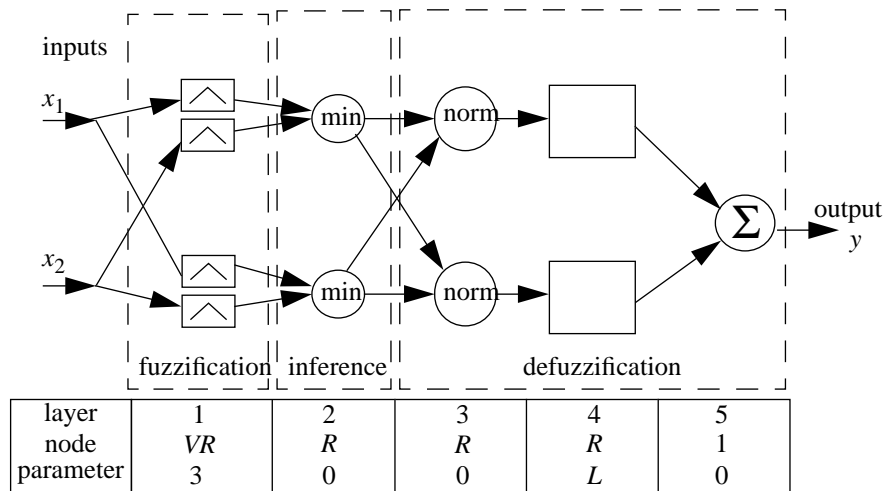


Figure 5.2 The basic structure of the FSOM. The table at bottom shows the number of nodes and the number of parameters of each node at each layer. V is the number of input variables. L is the number of output variables and R is the number of rules.

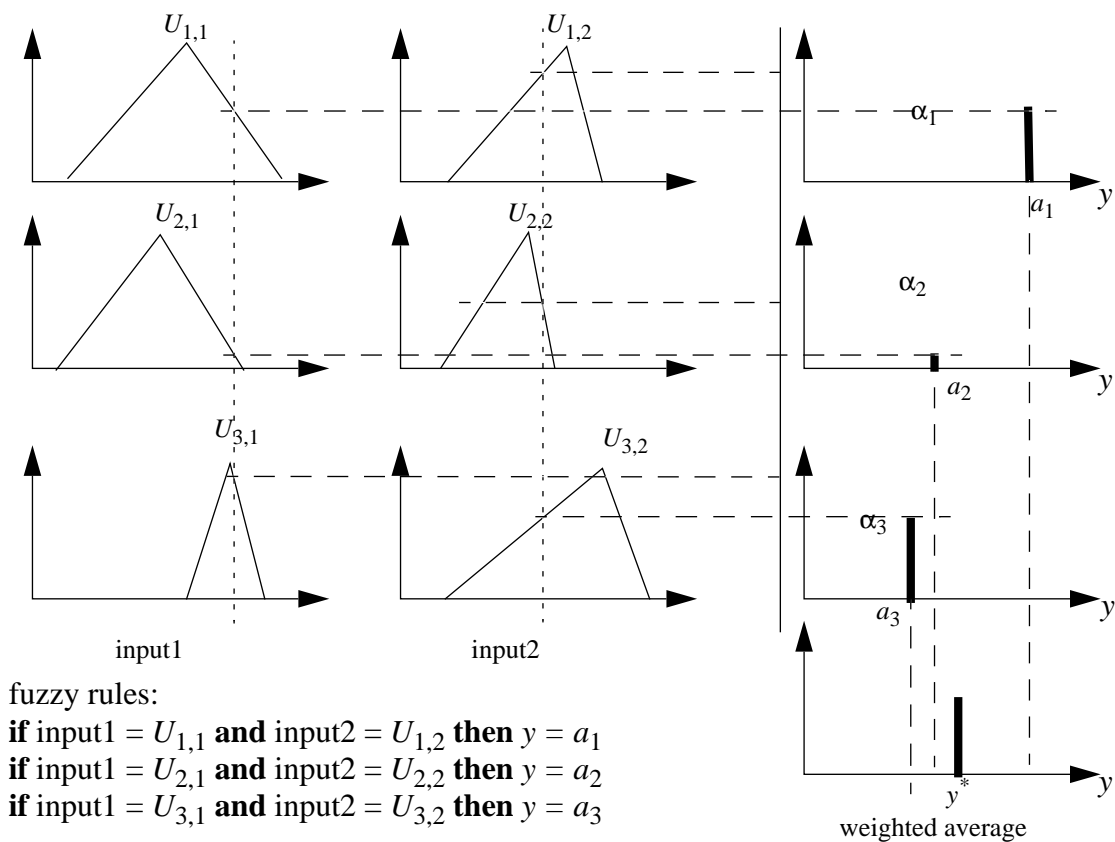


Figure 5.3 Operation principle of the FSOM.

5.2 Learning

Although the structure of the FSOM implements the fuzzy inference, it still reminds very much the structure of the original SOM. The learning idea of the FSOM is to use a modified Kohonen's LVQ2.1 learning rules to tune the membership functions. The learning scheme is supervised. That is, the system is trained based on training data which contains input-output data samples.

Tuning of fuzzy sets

In the FSOM, the antecedent membership functions are defined by three parameters and the consequent singleton by one parameter. Figure 5.4 illustrates the modified LVQ2.1 learning scheme. Three updating laws are used. Two updating laws modify the centers and the spreads of the antecedent fuzzy sets. The third updating law modifies the values of the output singletons.

Updating laws for spreads

The modified LVQ2.1 is also based on the concept of the window. One of the spread of a fuzzy set is moved, when at least two fuzzy rules fire simultaneously. The window, illustrated in Fig. 5.5, is simply defined as the overlapping area of the two most firing fuzzy rules. The *winner* rule w is firing most and the rule r is the *first runner-up* (firing second).

When the input sample falls into the window, one spread of the first runner-up rule is chosen to be updated. The spread $s_{r,k}$ is found out by computing the distances between the winner

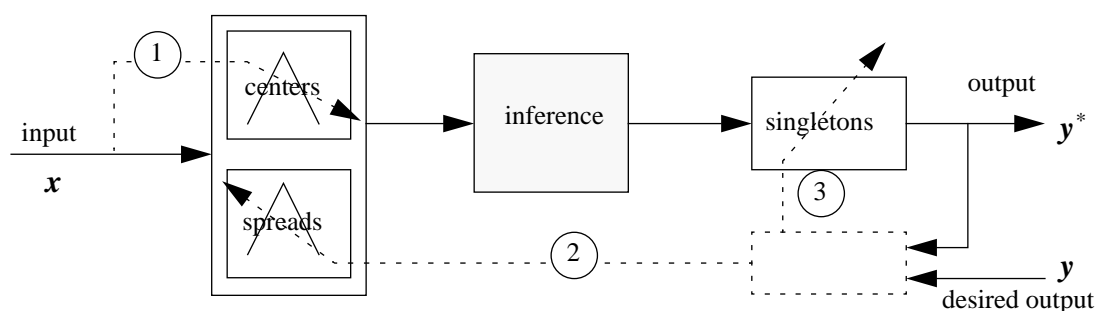


Figure 5.4 The learning of the FSOM. The centers and the singletons are updated when merely one fuzzy rule fires. The spreads are updated when at least two fuzzy sets fire simultaneously.

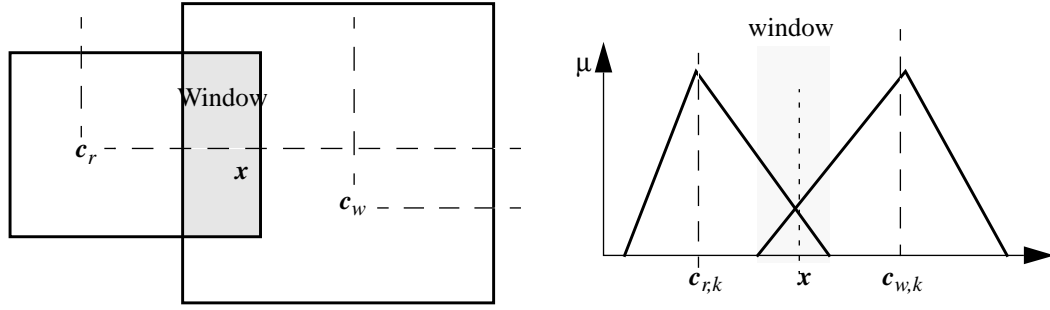


Figure 5.5 The window is defined as a overlapping area of the most firing winner rule w and the first runner-up rule r . The spread for the updating is selected from the input dimension where the center of rules have the largest distance.

and the first runner-up rule in every input dimensions and by choosing the input dimension and its spread where the distance between the winner and the first runner-up is the largest according to

$$|c_{w,k} - c_{r,k}| = \max_j \{|c_{w,j} - c_{r,j}|\}, \quad (5.6)$$

where $c_{w,j}$ and $c_{r,j}$ for $j = 1, 2, \dots, n$ are the center of the winner rule and the first runner-up, respectively. Recently, Vuorimaa [65, 48] has also introduced more sophisticated criteria for the selection of the spread.

The actual updating is done by moving the spread $s_{r,k}$ (i.e., $sl_{r,k}$ or $sr_{r,k}$) either towards the center $c_{r,k}$ or the spread $s_{w,k}$ of the winner rule w , as shown in Fig. 5.6. This either increases or decreases the influence of the first runner-up rule in the output of the FSOM. The updating is done according to

$$\begin{aligned} s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [c_{w,k}(t) - s_{r,k}(t)], & \text{sgn}(y - y^*) &= \text{sgn}(a_r - a_w), \\ s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [s_{w,k}(t) - s_{r,k}(t)], & \text{otherwise,} \end{aligned} \quad (5.7)$$

where $c_{w,k}$, $c_{r,k}$ are the centers and $s_{w,k}$, $s_{r,k}$ are the spread of the fuzzy rules. The learning rate of fuzzy sets g_U is chosen so that $1 > g_U \geq 0$. The variable y is the desired output of the training data set, y^* is the output of FSOM (5.5), a_w is the output of the winner rule, and a_r is the output of the first runner-up rule. The signum operator $\text{sgn}(x)$ is defined as

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0. \end{cases} \quad (5.8)$$

If the default rule is the first runner-up rule, the spreads or centers cannot be updated. Therefore, a slightly modified learning law has to be used. Now, the spread of the normal rule is chosen to be updated instead of the default rule according to

$$\mu_{U_{r,k}}(x_k) = \min_j \{\mu_{U_{r,j}}(x_j)\}. \quad (5.9)$$

Once again, the idea is to increase or decrease the influence of the normal rule. Thus, the spread $s_{r,k}$ is moved either away or towards the center $c_{r,k}$ of the same fuzzy set as follows

$$\begin{aligned} s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [c_{r,k}(t) - s_{r,k}(t)], & \text{sgn}(y - y^*) &= \text{sgn}(a_r - a_0), \\ s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [c_{r,k}(t) - s_{r,k}(t)], & \text{otherwise,} \end{aligned} \quad (5.10)$$

where a_0 is the output value of the default rule.

Updating laws for centers and singletons

When only one fuzzy rule fires, the centers and the singletons are updated. The centers c_w of the winner rule are updated towards the input sample x according to



Figure 5.6 The modified LVQ2.1 updating laws. On the left, the spread $s_{r,k}$, on the same side of the center $c_{r,k}$ as the input x_k , is moved either towards the center $c_{w,k}$ or the spread $s_{w,k}$ of the winner rule w . On the right, the centers $c_{w,j}$ of the winner rule w are moved towards the corresponding inputs x_j .

$$\mathbf{c}_w(t+1) = \mathbf{c}_w(t) + g_{U,w}(t) [\mathbf{x}(t) - \mathbf{c}_w(t)]. \quad (5.11)$$

The consequent singletons of the fuzzy rules are also updated, when only one fuzzy rule fires according to

$$a_w(t+1) = a_w(t) + g_{a,w}(t) \alpha_w(t) [y - y^*], \quad (5.12)$$

where $g_a(t)$ is the learning rate of the singletons ($1 > g_a(t) \geq 0$) and $\alpha_w(t)$ is the firing strength of the winner rule. In this case, also the default rule may be the winner rule.

5.3 Generation of fuzzy rules

Before the fuzzy sets can be updated, the fuzzy rules have to exist. The construction of the fuzzy rules can be done at least in two different ways. Either the fuzzy rules are defined in the first phase of the training, or the fuzzy rules are constructed during learning procedure based on self-generating heuristics.

Self-organizing rule generation method

The self-organizing rule generation method represents the first case. It consists of three separate stages. In the first stage, the centers of the fuzzy sets are self-organized using the standard SOM algorithm. In the second stage, the fuzzy sets are formed around the centers. Finally, the fuzzy sets are tuned by the modified LVQ2.1, as presented in previous section.

Stage 1. Self-organization of the centers of the fuzzy rules.

First, the number of the fuzzy rules has to be determined. After that, the centers of the fuzzy rules are self-organized by the unsupervised SOM algorithm.

Stage 2. Forming of fuzzy sets.

In the second stage, the fuzzy sets of fuzzy rules are formed around each center vector \mathbf{c}_i . The left sl_i and right spreads sr_i are given a constant width wi_0 . The spreads are computed according to

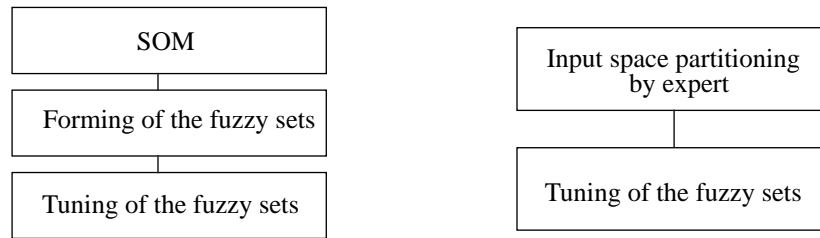


Figure 5.7 Self-organizing and expert rule generation and learning.

$$c_i - sl_i = sr_i - c_i = wi_0. \quad (5.13)$$

Stage 3. Tuning of fuzzy sets.

After the fuzzy sets and fuzzy rules are initialized, the fuzzy sets are tuned by using the modified LVQ2.1 learning.

Expert rule generation

The input space partitioning to fuzzy rules can also be done by expert. Of course, this requires knowledge about the behaviour of the modelled system. In addition, the capabilities of the FSOM should be known.

Self-generation

When the original scheme is used, the structure of the rule base can not change during learning procedure. That usually restricts the approximation accuracy, or can lead to use of many unnecessary fuzzy rules. Thus, the self-generation (self-construction) of the fuzzy rules is a convenient property. It makes possible to create new fuzzy rules only when they are needed. The self-generating procedure proposed by Vuorimaa [63] uses three heuristic rules to optimize the placement of the fuzzy rules:

1. The errors are removed in descending order.
2. New fuzzy rule is not added before the existing fuzzy rules have been tuned.
3. The existing fuzzy rules must not be disturbed.

The new fuzzy rules are added only when the current fuzzy rules perform poorly. The first heuristic rule requires that the fuzzy rules are added one by one to minimize the errors in descending order between the outputs of the FSOM and the training set. For this purpose, the largest error through the training set is first searched.

It is sensible to add new rules only, when the already existing fuzzy rules can not remove the current error. Thus, the second heuristic rule requires that the already existing fuzzy rules have to be stabilized. Hence, we must check whether there is an existing fuzzy rule which tries to remove the current error. This can be done by measuring how much each input sample overlaps with the already existing fuzzy rules. The overlapping of fuzzy rules is computed according to

$$\text{overlap} = \min_j \{ \text{overlap}_j \}, \quad (5.14)$$

where

$$\begin{aligned} \text{overlap}_j &= \frac{c_{i,j} - sl_{i,j}}{c_{i,j} - x_j}, & x_j \leq c_{i,j}, \\ \text{overlap}_j &= \frac{c_{i,j} - sr_{i,j}}{c_{i,j} - x_j}, & x_j > c_{i,j}. \end{aligned} \quad (5.15)$$

The older the already existing fuzzy rules are, the more the new fuzzy rule can overlap with them. The learning coefficients g_U are natural measures for the age of the fuzzy sets, since they decrease monotonously during the learning procedure. Hence, the second heuristic rule can be expressed as

$$\text{overlap} < \eta_1 g_{U,i}(0) / g_{U,i}(t), \quad (5.16)$$

where η_1 is a constant. A suitable value for $\eta_1 = 0.75$.

If the previous rules are valid, the new rules is added to the neuro-fuzzy system. The new rule n is initialized according to

$$\begin{aligned} \mathbf{c}_n &= \mathbf{x}, \\ \mathbf{a}_n &= \mathbf{y}. \end{aligned} \quad (5.17)$$

The third heuristic rule requires that the new fuzzy rule does not disturb the already existing fuzzy rules. Because of this, the width of the spreads must be limited according to

$$c_{n,k} - sl_{n,k} = \min \{ \eta_2 |c_{n,k} - c_{i,k}|, w_k \} \quad (5.18)$$

$$sr_{n,k} - c_{n,k} = \min \{ \eta_2 |c_{i,k} - c_{n,k}|, w_k \}$$

where

$$|c_{i,k} - c_{n,k}| = \max_j \{ |c_{i,j} - c_{n,j}| \} \quad (5.19)$$

and w_k is the default width of the spread. $\eta_2 = 0.75$ is a suitable value.

The training of the neuro-fuzzy system is started with a default rule and the normal rules are added according to laws during the training procedure. The initial output value of default rule is usually the mean of the output values in the training set. The heuristic rules can be applied and new fuzzy rules added after every sufficient training sample, but more preferable way is to evaluate and add new rules after each training epoch (one cycle through the training set).

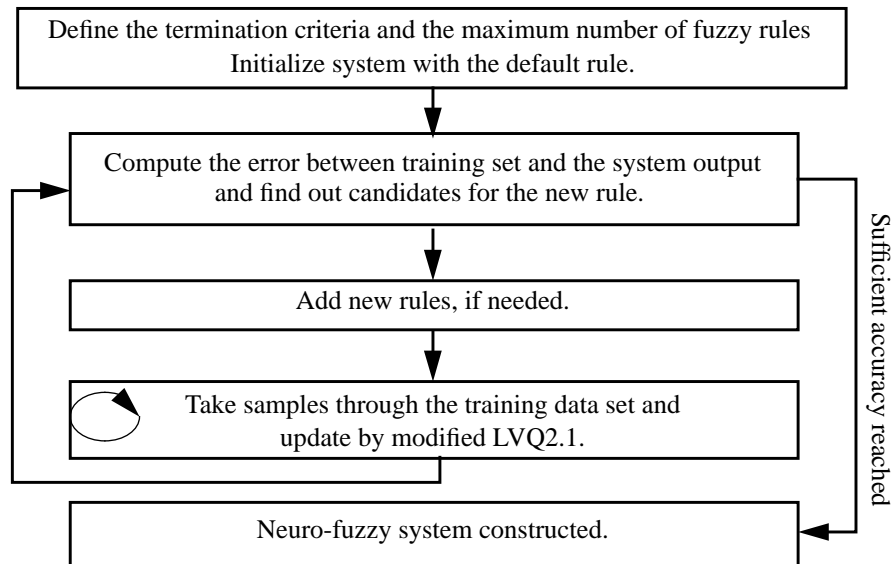


Figure 5.8 The self-generation procedure.

5.4 FSOM using Sugeno's fuzzy rules

Takagi and Sugeno [57] have introduced a type of fuzzy rules which have a functional consequence part instead of a fuzzy set or a singleton. The operation of the fuzzy system using Sugeno's fuzzy rules is illustrated in Fig. 5.9. The fuzzy rule maps a fuzzy input subspace linearly to the output space. Jang [22] has also used Sugeno's fuzzy rules in his neuro-fuzzy system. In this thesis, a FSOM using Sugeno's fuzzy rules was developed. The structure and learning of the functional FSOM [48] are very similar to the original FSOM.

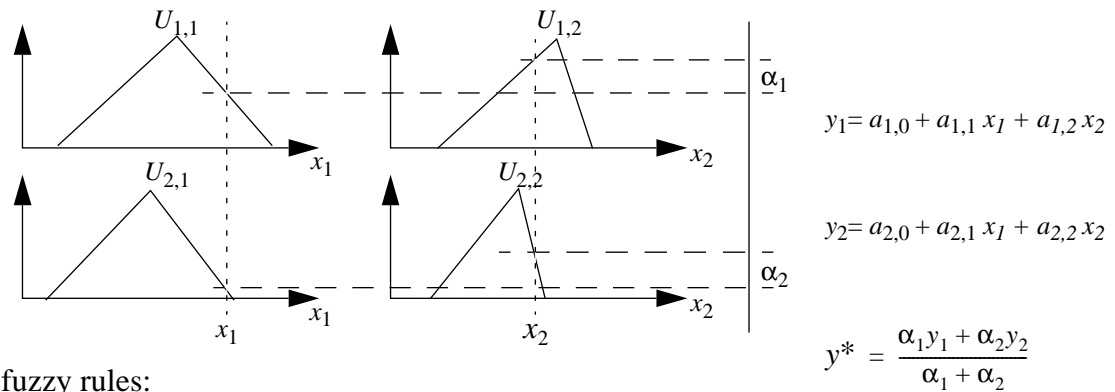
The fuzzy rules of Sugeno's type have the form:

$$\begin{aligned} &\text{if } x_1 \text{ is } U_{i,1} \text{ and } \dots \text{ and } x_n \text{ is } U_{i,n} \\ &\text{then } y \text{ is } a_{i,0} + a_{i,1}x_1 + \dots + a_{i,n}x_n, \end{aligned} \quad (5.20)$$

where a_j for $j = 0, 1, \dots, n$ are consequent parameters.

The output y^* of the FSOM is computed as a weighted average of the linear output functions, where the firing strengths α_i act as the weights according to

$$y^* = \frac{\sum_{i=0}^m \alpha_i (a_{i,0} + a_{i,1}x_1 + \dots + a_{i,n}x_n)}{\sum_{i=0}^m \alpha_i}, \quad (5.21)$$



fuzzy rules:

if $x_1 = U_{1,1}$ and $x_2 = U_{1,2}$ then $y = f(x_1, x_2)$.

if $x_1 = U_{2,1}$ and $x_2 = U_{2,2}$ then $y = f(x_1, x_2)$.

Figure 5.9 Operation of the FSOM using Sugeno's fuzzy rules.

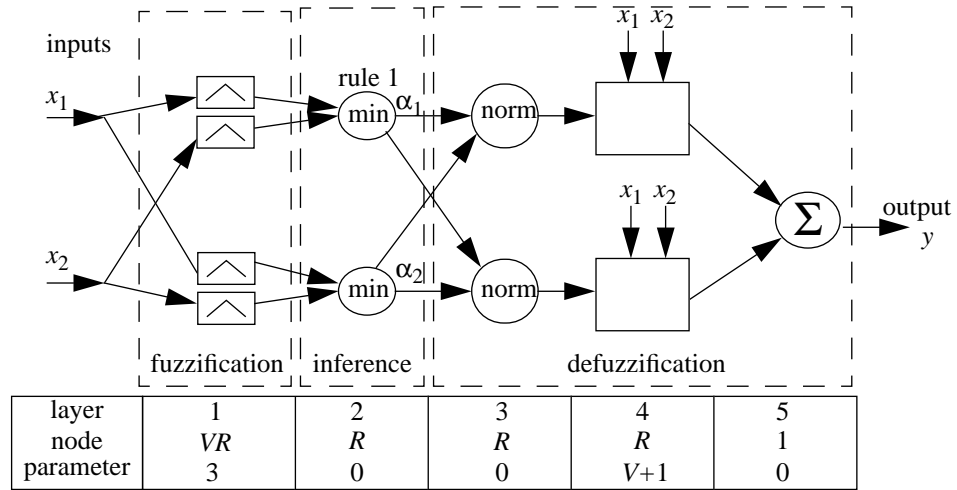


Figure 5.10 Basic structure of FSOM using Sugeno's fuzzy rules without default rule. The table at bottom shows the number of nodes and the number of parameters of each node at each layer. V is the number of input variables. R is the number of rules.

where m is the number of the fuzzy rules. The network structure which realizes the computation of the FSOM using Sugeno's fuzzy rules is shown in Fig. 5.10.

Tuning of consequent parameters

The learning procedure have two phases (Table 4). In the first phase, the antecedent parameters are fixed and the consequent parameters of the fuzzy rules are updated by using gradient descent algorithm. First, the input data vector is fed into the system, and the firing strengths of the fuzzy rules are computed according to (5.2), (5.3) and (5.4). The weight vector of each firing rule (normal and default rules)

$$\mathbf{a}_f = [a_{f,0}, a_{f,1}, \dots, a_{f,n}]^T \tag{5.22}$$

is updated according to

$$\mathbf{a}_f(t+1) = \mathbf{a}_f(t) + g_{o,f}(t) \alpha_f [y^*(t) - y(t)] [1, \mathbf{x}(t)]^T, \tag{5.23}$$

where $g_o(t)$ is the decreasing learning rate so that $1 > g_o(t) \geq 0$. α_f is the firing strength of the rule f .

Table 4: Two passes of the learning.

	antecedent parameters	consequent parameters
first pass	fixed	LMS
second pass	modified LVQ	fixed

Tuning of fuzzy sets

The antecedent parameters are tuned using modified LVQ2.1. First, the spread of the first runner-up rule is chosen according to

$$win_k = \min_j \{win_j\}, \quad (5.24)$$

where

$$win_j = \frac{\min \{sr_{w,j}, sr_{r,j}\} - \max \{sl_{w,j}, sl_{r,j}\}}{|c_{w,j} - c_{r,j}|}, \quad (5.25)$$

where win_j is the relative width of the window, and dimension win_k has the smallest relative width. After that, the chosen spread $s_{r,k}$ is updated according to

$$\begin{aligned} s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [c_{w,k}(t) - s_{r,k}(t)], & \text{sgn}(y - y^*) &= \text{sgn}(y_r - y_w), \\ s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [s_{w,k}(t) - s_{r,k}(t)], & \text{otherwise,} \end{aligned} \quad (5.26)$$

where $s_{w,k}$ is the spread of winner rule, $c_{r,k}$ and $c_{w,k}$ are the centers of the first runner-up and winner rule, respectively. y_w and y_r are the output values computed independently for each rule.

The learning law (5.26) can be employed when either one of the two most firing rules is the default rule. Then, the normal rule is always the first runner-up rule, and one of its spreads is updated according to

$$\begin{aligned}
s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [c_{r,k}(t) - s_{r,k}(t)], & \text{sgn}(y - y^*) &= \text{sgn}(y_r - y_0), \\
s_{r,k}(t+1) &= s_{r,k}(t) - g_{U,r}(t) [c_{r,k}(t) - s_{r,k}(t)], & & \text{otherwise.}
\end{aligned}
\tag{5.27}$$

Tuning of centers

The centers of the fuzzy sets are updated when only one normal fuzzy rule fires. The winner center \mathbf{c}_w is moved towards the input sample. The updating is done according to

$$\mathbf{c}_w(t+1) = \mathbf{c}_w(t) + g_{U,w}(t) \alpha_w(t) [\mathbf{x}(t) - \mathbf{c}_w(t)].
\tag{5.28}$$

5.5 FSOM simulation studies

Vuorimaa [61, 62, 63] has used a simple nonlinear function approximation problem to verify the learning algorithms of the FSOM. This learning problem is not very realistic, since the training data is uniformly distributed and the samples are noiseless. In this experimental study, a more realistic problem is considered for the self-generating FSOM.

The general study of learning characteristics is an extremely difficult or just impossible task. First of all, the tuning and construction of the self-generating system are done simultaneously during learning. Hence, it is very hard to evaluate whether the convergence is the effect of tuning of fuzzy sets or the addition of a new fuzzy rule. Secondly, the FSOM has several parameters, such as learning rates, default rule parameters, which can not be estimated analytically, but their values have to be obtained experimentally.

Simulation examples

The test function (Fig. 5.11) was the same as used by Vuorimaa [61, 62, 63] and Jang [22], a nonlinear sinc function

$$z = \text{sinc}(x_1, x_2) = \frac{\sin(x_1)}{x_1} \frac{\sin(x_2)}{x_2}.
\tag{5.29}$$

First, four data sets were generated. Two data sets were collected uniformly such that

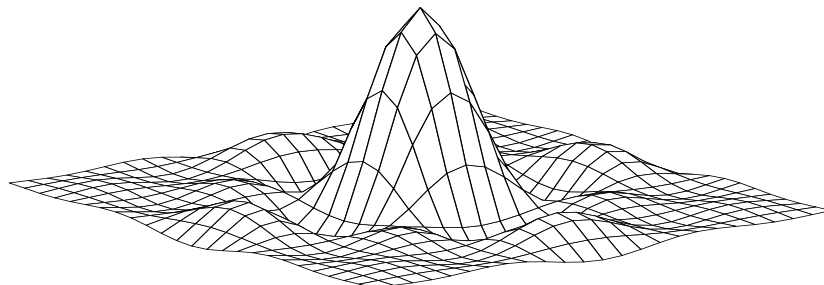


Figure 5.11 Sinc function.

$x_1, x_2 \in [-10, 10]$. Other two sets were formed by randomly selected input samples and their outputs (Table 5). The outputs z were ranged between $[-0.21, 1]$.

Next, the self-generating FSOM was used as function approximator. Each data set was used as training set and validation set. To make possible some kind of comparison between simulations, the FSOM was trained during 50 training epochs through the training data set. The initial fuzzy set width parameter $w_i = 2.5$ and initial learning parameters are $g_U = 0.06$, $g_a = 0.1$, $g_{a0} = 0.01$, $\beta_2 = 0.7$, and $\beta_1 = 0.3$.

Table 5: Data sets used in simulations.

no	data set
1	uniform, 225 samples
2	uniform, 625 samples
3	arbitrary, 225 samples
4	arbitrary, 625 samples

The validation of the system was evaluated using the training data set and the other sets. As an error measure between output of the neuro-fuzzy system y_i^* and data set output y_i , root-mean-square (RMS) error

$$E_{rms} = \sqrt{\frac{1}{n} \sum_{i=1}^n [y_i - y_i^*]^2} \quad (5.30)$$

is used. Table 6 shows the final accuracies of the neuro-fuzzy system, the number of the fuzzy rules and the accuracies on validation data sets. The result when training set 1 was used is much better than Vuorimaa's result 0.0103 [63]. However, the problem of the our better result is the decreased generalization. The system is dedicated to single training samples and the accuracy in the rest of input space is suffered. The learning using randomly chosen data samples produces much worse results, because the training data does not necessary contain sufficient information about the center peak of the sinc function.

Table 6: Final RMS errors of the self-generating FSOM and the number of the fuzzy rules in parantheses.

		training data set			
		1	2	3	4
validation set	1	0.0088 (25)	0.0437	0.0473	0.0478
	2	0.0456	0.0221 (21)	0.0420	0.0424
	3	0.0555	0.0329	0.0253 (15)	0.0426
	4	0.0524	0.0336	0.0454	0.0360 (15)

Fig. 5.12 shows how the fuzzy rules are located in the input space. Because of the symmetrical nature of sinc function, the fuzzy rules are also placed near symmetrically. The fast convergence of the system in the beginning of the training is mainly caused by the equalization of the peak of the sinc function in the origin.

Product operator

For curiosity, the previous tests where done using the product union operator replacing minimum operator. The main idea was to test up whether the product operator produces better generalization. When the product union operator is used, the firing strengths of fuzzy rules are computed as follows

$$\alpha_i = \mu_{U_{i,1}}(x_1) \mu_{U_{i,2}}(x_2) \dots \mu_{U_{i,n}}(x_n). \quad (5.31)$$

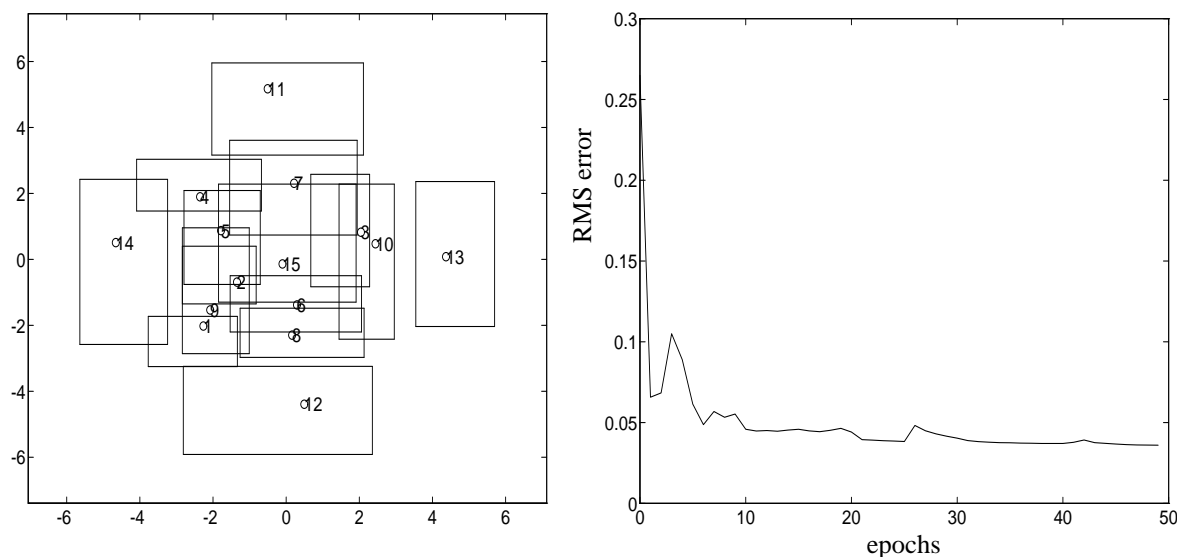


Figure 5.12 Fuzzy sets and convergence. On the left, the boxes are boundaries of the fuzzy sets (minimum operator, arbitrary distribution, 625 samples). On the right, the convergence of the FSOM is shown.

Table 7 shows the RMS errors in the same training cases than in Table 6. The results seem to show better relative generalization capability. Nevertheless, further studies are needed to get more reliable results.

Table 7: Final RMS errors of the FSOM using product connective. The number of the fuzzy rules in parentheses.

		training data set			
		1	2	3	4
validation set	1	0.0174 (20)	0.0317	0.0444	0.0280
	2	0.0415	0.0237 (20)	0.0387	0.0271
	3	0.0512	0.0276	0.0263 (17)	0.0286
	4	0.0426	0.0305	0.0409	0.0190 (23)

Chapter 6

Simulation examples

In this chapter, the fuzzy self-organizing map is employed into realistic control example cases. In the identification case, the fuzzy model of the nonlinear water heating process is identified. In the second example, the FSOM is used as neuro-fuzzy controller, while the third case deals with fault diagnosis. The computer simulations have been performed using MATLAB software.

6.1 Identification of a heating process

A model is a very useful and compact way to summarize the knowledge about a system. The theory of linear system models is very sophisticated. It offers several analysis and control design methods. However, real systems are always nonlinear to some extent. Hence, the nonlinear model would provide a much more multi-sided possibility to describe a plant. On the other hand, the identification of the nonlinear system is much more complicated, and there are only few methods for system analysis and control design.

Unlike the fully mathematical nonlinear model, a *fuzzy model* is capable of describing the behaviour of the system in a linguistic form. There are two different ways how the fuzzy model can be obtained: from a prior knowledge (e.g., in terms of physical laws) or by experimentation on a plant. The latter one is called *system identification*. The construction of the fuzzy model is similar to a general system identification procedure. The typical identification procedure consists of four phases [69]:

1. Experimental planning and data acquisition.
2. Selection of model structure.
3. Parameter estimation.
4. Model validation.

The experimentation with industrial processes is often difficult and expensive. Thus, it is desirable that the measurements do not require any special input excitation signals. The main requirement of the input signal is to excite all the modes of the process sufficiently. In nonlinear identification, the amplitude contents have to be taken into account, in addition to frequency contents. Usually, the pseudo random (PR) -excitation signal is used.

The second phase is the most important and the most difficult one. The problem is to select the inputs and the outputs of the model, but also the function approximator. In a nonlinear case, a neural network or a neuro-fuzzy system are good, because of their capability to approximate nonlinear functions.

The parameter estimation can be formulated as an optimization problem, where the best parameters for a model are found. Often, the aim of the estimation is to minimize the sum of the error squares. In neural computation, this stage is usually called training of network.

Model validation forms the final stage of any identification procedure. After parameter estimation the performance of the model must be verified by a validity test. There are a number of well-established validity tests for linear systems which are usually based on correlation analysis. For nonlinear systems, Billings and Voon [6] have proposed some validity tests which are extensions of linear correlation analysis. Nevertheless, the fit of the model is usually verified by using simulations and comparing the results to measurements.

Nonlinear heating process

Our example case is a laboratory-scale water heating process in control engineering laboratory of Tampere University of Technology. It consists of an uninsulated tank (0.45 litre) and a 3 kW heating resistor, as shown in Fig. 6.1. The input flow q_{in} can be set with rotameter between 0 - 3.0 l/min, here it is 1.25 l/min. The output signal of the system is the temperature of the outlet water which is measured with a Pt-100 semiconductor transducer. The input of the system is the driving voltage of the thyristor. The time delay of the system is 15 seconds and the dominating time constant about 30 seconds. The nonlinearities of the process are

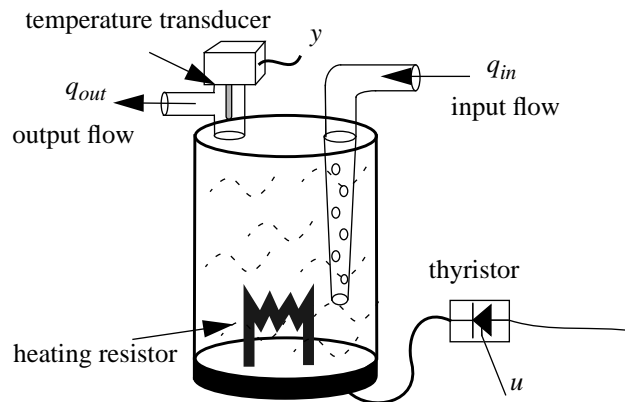


Figure 6.1 The water heating process.

mainly caused by the saturation of the thyristor.

The measurements of the system were made by Ruoppila [52]. Two separate data sets are available. They are recorded during an open loop experiment and they cover the whole operation range from 10 °C to 60 °C. The input excitation was a pseudo random (PR) signal, and the sampling interval was three seconds.

The first data set (3300 samples) are used in the parameter estimation, while the second set including 800 samples is used in the model validation. The training data between 2500 and 3000 sampling instant are shown in Fig. 6.2.

Linear ARX model

Before the fuzzy models of the system are identified, a linear ARX-model of the process is identified. If the noise of the measurements is stationary, the system can be represented by an ARX model having the simple form suggested in [53]:

$$\hat{y}(t) = -ay(t-1) + bu(t-d) \quad (6.1)$$

where $\hat{y}(t)$ is the prediction of the measurements $y(t)$ at discrete time instant t , $y(t-1)$ is the measurement in previous time instant, $u(t)$ is the input and d the delay. In the heating process, the delay is 5 sampling instants.

The goal of the parameter estimation is to minimize the mean squared error

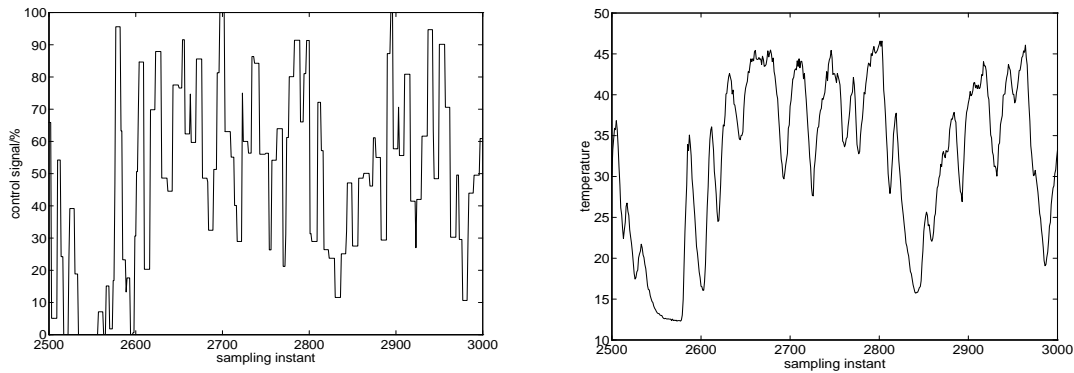


Figure 6.2 Snapshot of the excitation signal (on the left) and the temperature measurement (on the right).

$$V = \frac{1}{M} \sum_{i=1}^M [y_i(t) - \hat{y}_i(t)]^2, \quad (6.2)$$

where M is the number of data samples. Parameter estimation produced values $a = 0.9373$ and $b = 0.0401$ for the linear model. The error measure of the training set was $V = 0.977$ and the verification data set $V = 0.572$.

Nonlinear ARX models

The most simple stochastic nonlinear prediction model has the form

$$\hat{y}(t) = g(y(t-1), u(t-d), \Theta), \quad (6.3)$$

where $g(\cdot)$ is a nonlinear function and Θ is its parameter vector. This model is called nonlinear autoregressive with exogenous input (NARX) model.

Training of neuro-fuzzy systems

Three different self-generating FSOMs were used to form this prediction NARX model. Before training, the actual training data was extracted. The delay of 5 sampling instants was taken into consideration. No special scaling of input values was required. Two original self-generating FSOMs were used: one using minimum operator and another using product operator. The training procedures were started with only the default rule and new fuzzy rules were added to the neuro-fuzzy system during the learning process. Fig. 6.3 shows the convergences of the systems.

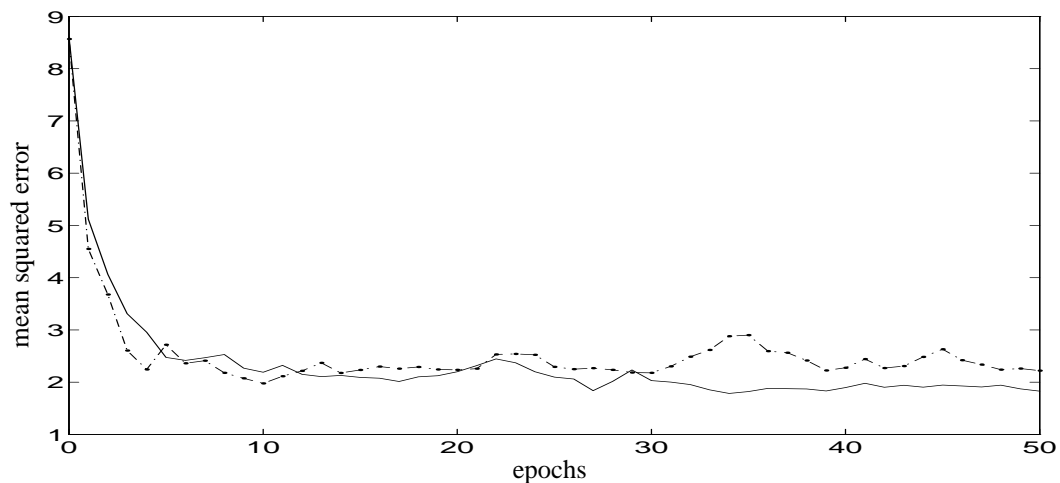


Figure 6.3 The convergences of the self-generating FSOM (solid line is with product operator, dash-dotted minimum).

The linear ARX model was used as a starting point of the FSOM using Sugeno's fuzzy rules. The consequence parameters of the new fuzzy rules were initialized with the parameters which were estimated for the ARX model. Fig. 6.4 shows how this system converged.

Results

In Table 8, the neuro-fuzzy models are compared to the linear ARX model and nonlinear ARX model constructed by a radial basis function network [53]. The FSOM using Sugeno's fuzzy rules produces very good results. The self-generating FSOMs do not perform very well. The reasons to this are that the training samples are noisy, and the self-generating method has originally been design for learning of noiseless systems. Now, new fuzzy rules are easily added in the locations of noisy samples. In this example case, the FSOM which employs product operator has again better generalization capability than the minimum operator FSOM.

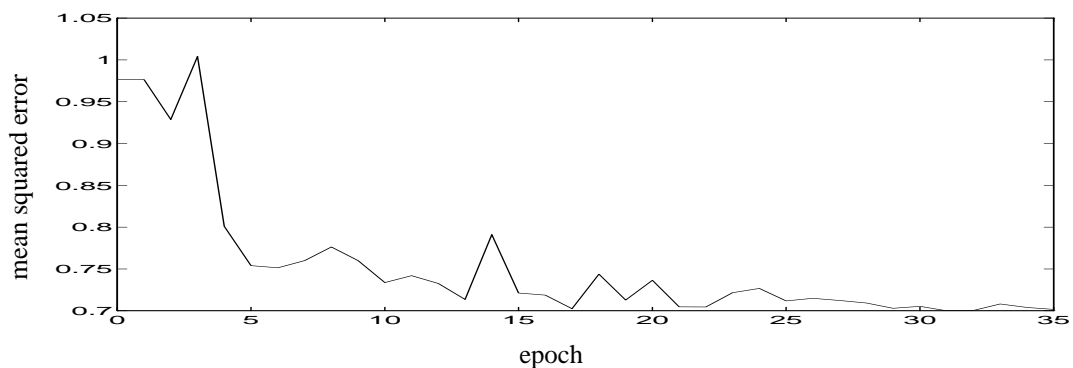


Figure 6.4 Convergence of the FSOM using Sugeno's fuzzy rules.

Table 8: Costs (6.2) for ARX and nonlinear ARX models.

Method	training set	Validation set	number of rules or units
ARX	0.9765	0.5716	—
Self-generating FSOM (minimum)	1.8543	1.9380	20
Self-generating FSOM (product)	2.2997	1.8213	20
FSOM Sugeno's fuzzy rules	0.7147	0.3619	10
RBF [53]	—	0.3675	8

6.2 Model-based neuro-fuzzy controller

During past several years, fuzzy control have emerged as one of the most active and fruitful areas for research in the applications of fuzzy logic [34]. Neural networks have also been used as controllers [19]. The neuro-fuzzy systems have been introduced to collect the strengths of neural networks and fuzzy logic. The neuro-fuzzy systems are to capable of learning nonlinearities and the knowledge of the system is in a human understandable form as linguistic fuzzy rules. So far, the main emphasis in the research has been on automatic design and fine-tuning of the membership functions used in fuzzy logic controllers through learning by neural networks.

Control design schemes

Control design schemes of the neuro-fuzzy systems are basically similar to the methods used with neural networks. The controller learning schemes can be divided into four different approaches:

1. Supervised learning.
2. Model-based learning.

3. Learning-based learning.
4. Reinforcement learning.

If the behaviour of the controller is known, the neuro-fuzzy controller can be constructed and tuned based on it. First, the training data can be obtained from numerical input-output data from an existing controller or the control actions of an human operator. After that, the neuro-fuzzy system is trained using supervised learning.

In model-based or *indirect* control design approach, the control action values are not readily available. but they must be figured out. The model of the process is used in the construction of the controller, as illustrated Fig. 6.6. The plant model can be fully mathematical, a neural network or a FSOM. It is important to notice that the model of the process makes it possible to try different control actions without that the state of the actual plant changes. In order to define which input-output behaviour of the controller is good, the some kind of supervisor or critic element is needed. Usually, the goal of the learning is to minimize the output error between desired output and the output of the process model.

The learning-based scheme is also called *direct* method [46]. In this scheme, the model of the plant is not constructed, but the construction and tuning of the controller proceeds while control actions are applied to the system. Because the plant changes its state after each control action, the learning can not be very fast.

The reinforcement learning differs from the direct scheme so that there is no supervisor to critically judge the chosen control action at each time step. The scheme uses an evaluative unit to predict indirectly the effects of control actions. In addition, a more challenging problem is that the reinforcement signal may only be available at a time long after a sequence of actions has occurred. Berenji and Khedkar [3, 4] have suggested so called GARIC (Generalized Approximate Reasoning based Intelligent Control) scheme, which employs a reinforcement learning technique in conjunction with a multilayer neural network model of a fuzzy

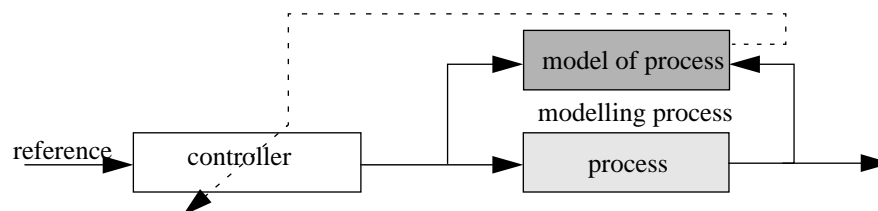


Figure 6.5 The model-based control scheme.

controller. Similarly, Lin *et al.* [39] uses a structure similar to the GARIC, but the evaluative unit is replaced by a fuzzy predictor.

Model-based approach for inverted pendulum problem

Vuorimaa [65] has used a model-based approach in the design of the FSOM based neuro-fuzzy controller for the truck backing-up problem [29]. As control design example, the neuro-fuzzy system is now employed as a state controller for an inverted pendulum problem.

Inverted pendulum problem

The inverted pendulum is a classic example of an inherently unstable system. This problem is well-studied using conventional methods, fuzzy systems and neural networks [1], but also some works using neuro-fuzzy systems have been done [2, 4, 21, 39]. The task involves a pendulum hinged to the top of a wheeled cart that travels along a track, as shown in Fig. 6.6. The cart and the pendulum are constrained to move within the vertical plane. The state at time instant k is specified by four real-valued variables: the pole angle θ , the angular velocity $\dot{\theta}$, the horizontal position and the velocity of the cart. The dynamics of the pole can be defined according to

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left[\frac{-F - m l \dot{\theta}^2 \sin\theta}{m_c + m} \right]}{l \left[\frac{4}{3} - \frac{m \cos^2\theta}{m_c + m} \right]}, \quad (6.4)$$

where $g = 9.8 \text{ m/s}^2$ is the acceleration due to gravity, F is the force applied to the cart (output of the controller), $m_c = 1.0 \text{ kg}$ is the mass of cart and $m = 0.1 \text{ kg}$ is the mass of pendulum, and $l = 0.5 \text{ m}$ is the distance from the pivot to the pendulum's center of mass.

The aim of the controller is to apply a force sequence such that the pendulum is balanced and the cart does not hit the edge of the track. In this study, a simplified version of this problem is studied [21], where the second condition is ignored.

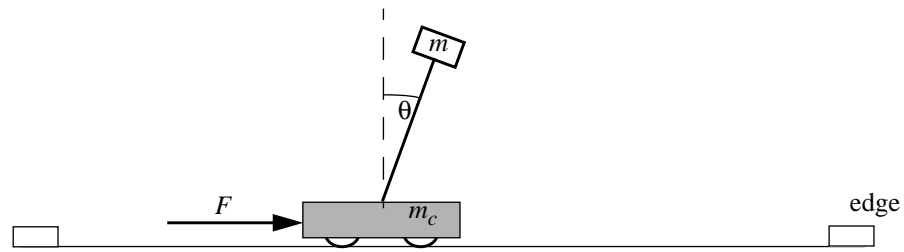


Figure 6.6 The inverted pendulum.

Neuro-fuzzy state controller

Let us use a discrete state-controller which realizes a control law

$$u(k) = f(\mathbf{x}(k), \Theta), \quad (6.5)$$

where $u(k)$ is the control action, f is the control function, $\mathbf{x}(k)$ is the state of the system and Θ is the parameter vector of the controller. In our neuro-fuzzy controller, the parameter set contains the centers, the spreads, and the singletons of the fuzzy rules and the number of the fuzzy rules.

Figure 6.7 shows the control system, which consists of the neuro-fuzzy controller and the inverted pendulum plant. Given state of the plant at the time instant k , the neuro-fuzzy controller will generate an input to the inverted pendulum plant and the plant will evolve to the next state at the time instant $k+1$.

Training of the neuro-fuzzy controller

The training data set in each training epoch was generated by simulating the control system from two initial state positions (10,0) and (-10,0) during 100 time steps using the fixed controller. The inverted pendulum system was simulated by the linear approximation prediction method and a time step 10 ms. The constraints on the variables were $-20^\circ \leq \theta \leq 20^\circ$ and

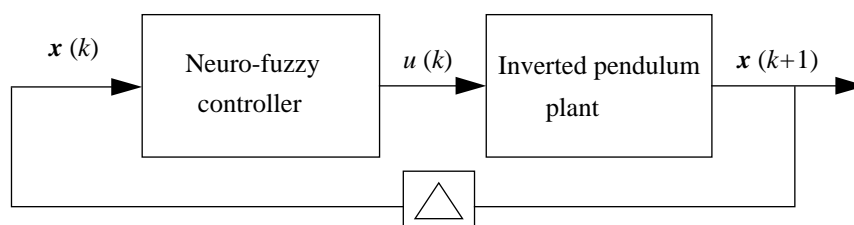


Figure 6.7 The control system.

$-50^\circ \leq \dot{\theta} \leq 50^\circ$. Consequently, the training data set contained desired input-output pairs of format

$$(\text{system state, desired angle}), \quad (6.6)$$

where the system state is a two-element vector which is composed of the angle and the angular velocity of the pole in each time step. In simulations, the desired trajectory was always zero. To achieve the control goal in a near-optimal manner, the objective of the training is to minimize the cost function

$$J = \sum_{k=2}^K \theta^2(k) + \lambda \sum_{k=0}^{K-2} u^2(k), \quad (6.7)$$

where the coefficient λ accounts the relative cost of control action.

The FSOM using Sugeno's fuzzy rules is now employed. The following procedure was used to tune the neuro-fuzzy controller:

- Step 1. The output of the FSOM y^* (control action u) for the actual input state is computed. After that, the costs for this action and the altered actions $y^+ = y^* + \Delta y$ and $y^- = y^* - \Delta y$ are evaluated, where Δa is a small change. After that, the consequent parameters are updated to the direction, which reduces the value of the cost J according to

$$\begin{aligned} \mathbf{a}_f(t+1) &= \mathbf{a}_f(t) - g_{o,f}(t) \alpha_f [1, \mathbf{x}(t)]^T, \quad \text{if } J_r(y^-) < J_w(y^+) \\ \mathbf{a}_f(t+1) &= \mathbf{a}_f(t) + g_{o,f}(t) \alpha_f [1, \mathbf{x}(t)]^T, \quad \text{otherwise,} \end{aligned} \quad (6.8)$$

- Step 2. If two normal fuzzy rules fire, the spread of the first runner-up is updated so that the cost function J reduces. The updating is done according to

$$\begin{aligned} s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [c_{w,k}(t) - s_{r,k}(t)], & \text{if } J_r > J_w, \\ s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [s_{w,k}(t) - s_{r,k}(t)], & \text{otherwise,} \end{aligned} \quad (6.9)$$

and the case of a normal rule and the default rule as follows

$$\begin{aligned}
s_{r,k}(t+1) &= s_{r,k}(t) + g_{U,r}(t) [c_{r,k}(t) - s_{r,k}(t)], & \text{if } J_r > J_0, \\
s_{r,k}(t+1) &= s_{r,k}(t) - g_{U,r}(t) [c_{r,k}(t) - s_{r,k}(t)], & \text{otherwise.}
\end{aligned} \tag{6.10}$$

If only one normal rule fires, the center of this rule is updated according to

$$\mathbf{c}_w(t+1) = \mathbf{c}_w(t) + g_U(t) \alpha_w [\mathbf{x}(t) - \mathbf{c}_w(t)]. \tag{6.11}$$

Initial and trained neuro-fuzzy controller

First, the initial parameters for the controller were set. The consequent parameters of the FSOM were all set at zero, which means that the control actions were zero initially. Figure 6.8 shows the control action surface initially and after training. The initial fuzzy if-then rules were

$$\begin{cases}
\text{if } \theta \text{ is } A_1 \text{ and } \dot{\theta} \text{ is } B_1 \text{ then } u = 0 \\
\text{if } \theta \text{ is } A_2 \text{ and } \dot{\theta} \text{ is } B_2 \text{ then } u = 0 \\
\text{if } \theta \text{ is } A_3 \text{ and } \dot{\theta} \text{ is } B_3 \text{ then } u = 0 \\
\text{if } \theta \text{ is } A_4 \text{ and } \dot{\theta} \text{ is } B_4 \text{ then } u = 0
\end{cases} \tag{6.12}$$

where A_i and B_i are linguistic labels characterized by triples (the left spread, the center, the right spread) as follows $A_1 = \{-30, -20, 5\}$, $A_2 = \{-5, 20, 30\}$, $A_3 = A_1$, $A_4 = A_2$, $B_1 = \{-75, -50, 25\}$, $B_2 = \{-25, 50, 75\}$, $B_3 = B_1$, $B_4 = B_2$.

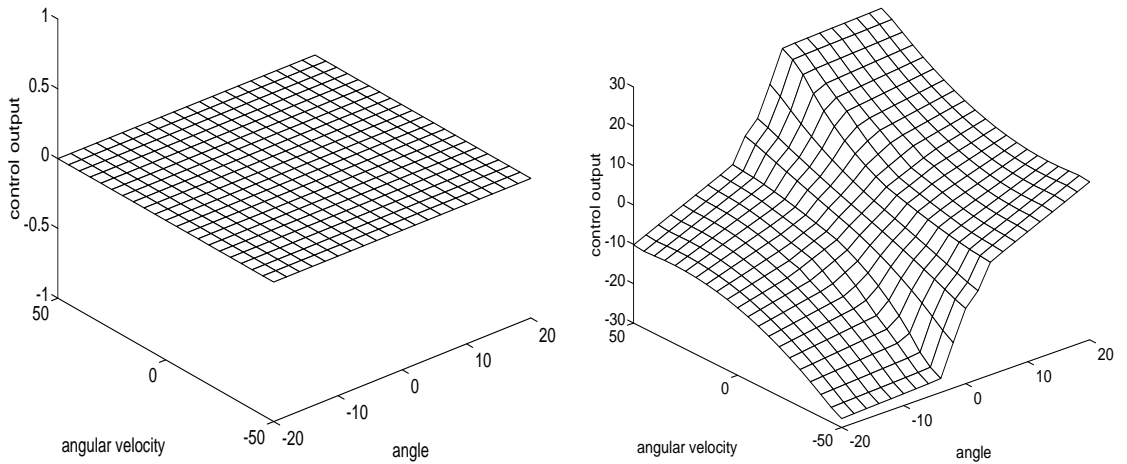


Figure 6.8 The initial and the final control action surface.

The final fuzzy if-then rules after training were

$$\left\{ \begin{array}{l} \text{if } \theta \text{ is } A_1 \text{ and } \dot{\theta} \text{ is } B_1 \text{ then } u = 0.051 \cdot \theta + 0.161 \cdot \dot{\theta} - 11.01 \\ \text{if } \theta \text{ is } A_2 \text{ and } \dot{\theta} \text{ is } B_2 \text{ then } u = 0.711 \cdot \theta + 0.062 \cdot \dot{\theta} - 0.974 \\ \text{if } \theta \text{ is } A_3 \text{ and } \dot{\theta} \text{ is } B_3 \text{ then } u = 0.711 \cdot \theta + 0.062 \cdot \dot{\theta} + 0.974 \\ \text{if } \theta \text{ is } A_4 \text{ and } \dot{\theta} \text{ is } B_4 \text{ then } u = 0.051 \cdot \theta + 0.161 \cdot \dot{\theta} + 11.01 \end{array} \right. \quad (6.13)$$

where A_i and B_i are linguistic labels characterized by triples (the left spread, the center, the right spread) as follows $A_1 = \{-30, -19.1, 4.04\}$, $A_2 = \{-4.04, 19.1, 30\}$, $A_3 = A_1$, $A_4 = A_2$, $B_1 = \{-75, -45, 44.05\}$, $B_3 = \{-44.05, 44, 75\}$, $B_2 = B_1$, $B_4 = B_3$.

Initially, the controller is not capable of balancing the pole into the top position. Figure 6.9 shows the trajectories of the pole under control of the final controller, and Fig. 6.10 shows the control action sequences applied to the plant. Visually, the results seems good. However, the convergence of the systems took very much time. That is mainly because the iterative process searching the correct outputs, but also because of the updating law of consequent parameters which is very slow.

In future development of the neuro-fuzzy tuning procedure, the updating of the consequent parameters seems to be essential. The simple gradient-descent have too slow convergence compared to the modied LVQ. Fortunately, there are many sophisticated methods of linear optimization available.

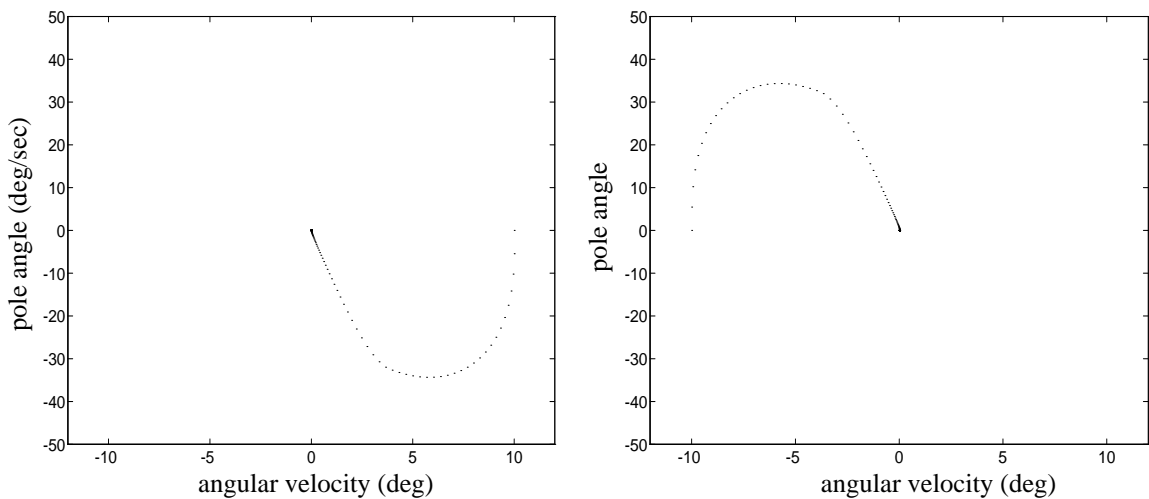


Figure 6.9 Trajectories of the pole from two initial positions (10,0) and (-10,0).

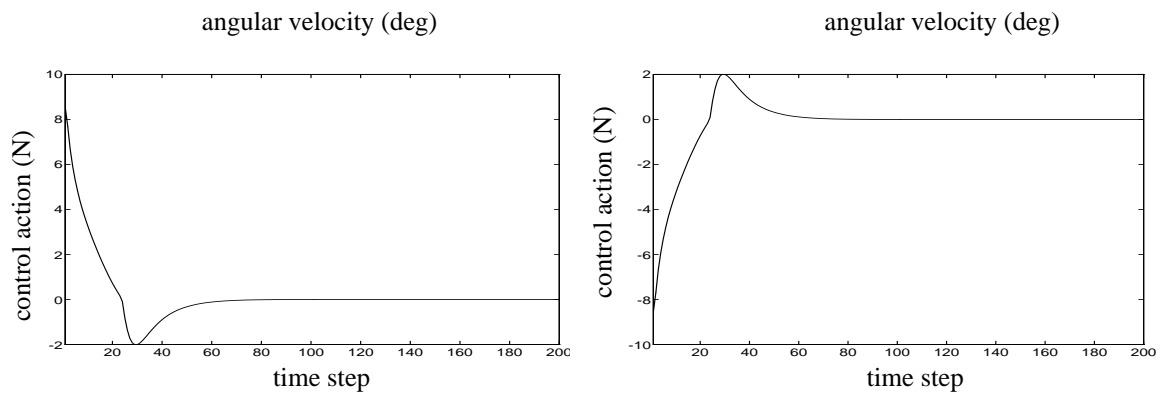


Figure 6.10 The control action sequences applied to the plant from two initial positions (10,0) and (-10,0).

6.3 Fault diagnosis of a reactor and a heat exchanger system

In the following simulation example, the FSOM is applied to fault diagnosis. The process under this study is a reactor and a heat exchanger process, studied by Sorsa *et al.* [54]. They have also presented the mathematical model of the process.

The significance of the fault detection has increased in process automation. The early fault detection and diagnosis is very important from the viewpoint of reducing manufacturing costs and plant safety. Nowadays, the controlled systems are so wide and complex that the operators are not able to know all the details of the system. Next some common fault diagnosis methods are described [54].

The limit checking of process measurements is the most conventional way used in fault detection. The new measurements are compared to the predetermined limit values and the alarm is given, if the limits are exceeded. This approach is very simple, but it has drawbacks. The limits have to be wide, specially when the measurements are noisy. This leads easily to a great number of simultaneous alarms.

In model-based fault diagnosis, the estimated states of the system are compared to the mathematical model of the system. The differences (residuals) between the states computed using the model and the estimated states are then used in the detection of the faults. Model-based methods consider that the exact linear process model is available. That is not always possible or economical.

Pattern recognition methods are suitable to solve the fault diagnosis problem, especially if the process model is not known. The idea of the pattern recognition system is to form a mapping from the measurements to the fault classes as shown in Fig. 6.10. Feedforward neural networks can be used for this kind of nonlinear mappings. The measurements are either fed directly to the pattern classifier or they are first preprocessed.

A reactor and a heat exchanger process

The example process consists of a heat exchanger and a continuously stirred tank reactor where an irreversible first order reaction $A \rightarrow B$ takes place. The reaction is catalytic and exothermic. The temperature of the reactor is controlled by pumping a part of the reactor out-

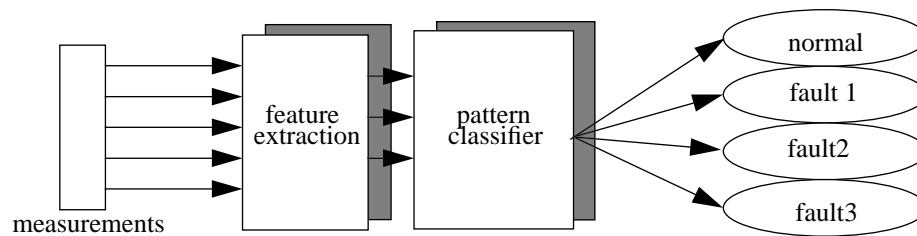


Figure 6.11 Pattern classifier fault diagnosis system.

let stream back to the reactor through the heat exchanger where the recycle flow is cooled by an external flow. PI controllers are used to keep the recycle flow rate, the level and the temperature of the reactor constant.

The process has nonlinear dynamics with both slow and fast components. In addition of this, the changes in the recycle flow rate affect strongly the reaction speed. The noise is also added to the measurement in simulation data. The noise varies from 0 - 10% of measurement regions. Altogether, fourteen variables are measured from the process as shown in Fig. 6.11. Ten fault situations and one normal situation are listed in Table 9.

Table 9: Fault descriptions.

no	description	no	description
0	normal operation	6	Deactivated catalyst
1	Input pipe partially blocked	7	Temperature control valve stuck high
2	Recycle pipe partially blocked	8	Leak flow in reactor
3	Input concentration of A high	9	Recycle flowmeter stuck high
4	Recycle flow setpoint high	10	Malfunction in pump
5	Fouled heat exchanger		

FSOM classifier

A set of 440 samples was available containing 10 samples per fault class. First, the four training data sets were generated by taking randomly 330 samples from the whole set. The rest of samples were used as validation data sets. Before training, the data sets were extracted to the form which is capable to train the FSOM. The fault numbers were encoded to a 11 length

vector. The component having class number was set to one and all other components was remained zero.

The multiple-input multiple-output FSOM was used. Vuorimaa and Jukarainen [64, 66] have also used this version as a pattern classifier. The rule construction of this classifier version is slightly different from the self-generation method. The locations for new rules are searched among the samples in the same class. The vector median of each class is chosen as the initial centers, and the spreads cover the whole class samples.

The learning of the neuro-fuzzy classifier was started with a default rule ($g_{a0} = 0.1$, $\beta_2 = 0.7$ and $\beta_1 = 0.3$). The default rule was associated to the normal state of the process and the other fuzzy rules were added to the classifier epoch by epoch where they are mostly needed (the greatest number of misclassifications). The learning rates were $g_U = 0.05$ and $g_a = 0$. Four FSOM classifiers were self-generated and tuned using four separate training data sets.

Results

The convergences of FSOM classifiers on training data are shown in Fig. 6.12. The final classifiers employed 10 fuzzy rules and one default rule. This is the minimum number of the fuzzy rules needed to solve this classification problem. Table 10 shows the number of misclassifications on separate validation data which includes 110 samples. Sorsa [54] used three different neural networks in the same problem. The accuracies of these networks were good, between one and ten misclassifications on validation data. However, the FSOM classifiers produce much better results.

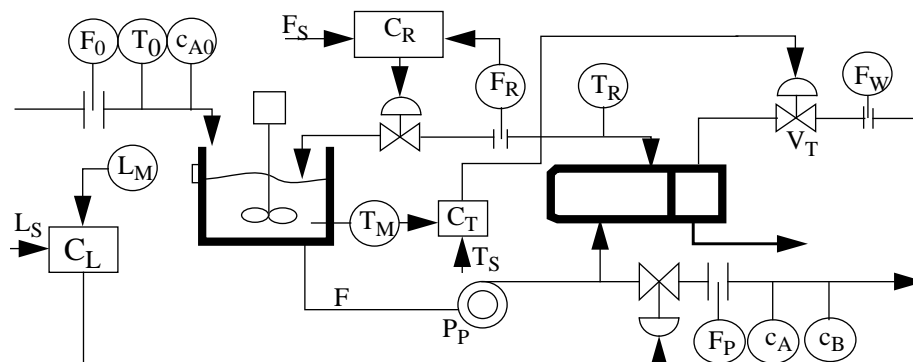


Figure 6.12 The example process and the measurements.

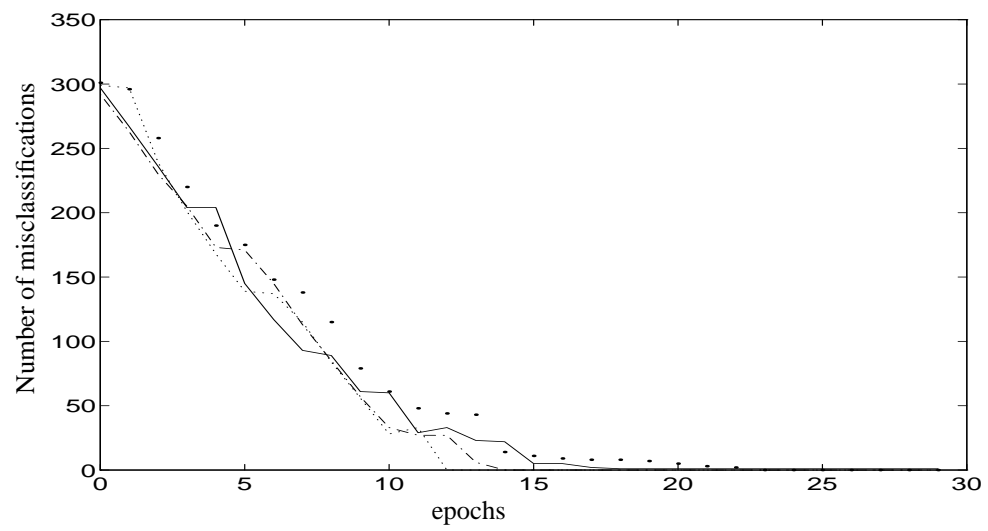


Figure 6.13 The convergence of four separate FSOM classifiers on training set.

Chapter 7

Discussion

One of the main goals of a neuro-fuzzy system is to combine the linguistic representation of fuzzy systems and learning capability of neural networks. The fundamental problem of the FSOM is the complexity of the knowledge base. Each fuzzy rule has its own antecedent fuzzy sets defined for every input variable. That implies it is very hard to produce any sensible linguistic rule expressions by using FSOM. The weaknesses of the structure can not easily be remedied. Two different ways to solve this problem can be used. In the first one, the whole structure should be changed in the way, that antecedent membership functions are defined independently for each input dimension. The modified LVQ learning can still be employed for this kind of structure. This improvement produces fuzzy rules which can be understood. In addition, it also reduces the computational complexity. On the other hand, the structure of the FSOM could be simplified significantly after the system is constructed and tuned.

The self-generating method has also several problems. If the training data is noisy or it has to be determined through an iterative process, the method seems to fail to find good rule allocations. In addition, the self-generating method seems to be too simple to find excellent allocations of fuzzy rules.

The self-generating method is closely related to the structure of the fuzzy rule structure. For the neuro-fuzzy system proposed above, some very promising structure identification methods are suggested [55]. In addition, some interesting methods to remove unnecessary or redundant inputs and fuzzy rules are presented [17, 44].

Unfortunately any proofs of convergence of the modified LVQ learning scheme has not

been introduced. So far, there are no analytical methods to find optimal values for learning rates. The learning and construction parameters have to be found out experimentally. This may be very slow process. The same problem appears with original Kohonen's learning algorithms: the convergence of the SOM and LVQ algorithms have not been proven.

At the present, there is still possibilities to develop the learning rules. For example, a problem concerns the updating laws of the centers and the singletons, equations (5.11) and (5.12). The outputs of the fuzzy sets are updated, when only one fuzzy set fires. However, the self-generation procedure often adds new fuzzy rules so that the center of the fuzzy sets or singletons are never updated, because the new rule is inside the other fuzzy rules.

Chapter 8

Summary

In this thesis, the neuro-fuzzy systems were studied starting from the basics of fuzzy logic and neural networks. Among the neuro-fuzzy systems the main interest was focused on the fuzzy self-organizing map. Its structure and learning algorithms were described. A new version of the FSOM was also developed called the FSOM using Sugeno's fuzzy rules.

The modified LVQ learning law can be considered as the most exciting method of the FSOM. Kohonen's learning rules tune the membership functions of the FSOM very efficiently compared to the backpropagation algorithm. The FSOM learning scheme has usually fast convergence, unfortunately it is strongly dependent on the initial parameter values.

In the FSOM, the linguistic variables are defined for each fuzzy rules, independently. Hence, the linguistic representation of the FSOM is not very understandable. Also many other neuro-fuzzy systems suffers from this. At present, the structure of the FSOM is not optimized after the learning procedure. It is obvious that the most of the fuzzy rules do not always need a fuzzy set for every input variable. Hence, both the structural and linguistic complexity could be reduced by refining the rules.

So far, the use of the FSOM in pattern classification tasks has been studied most. The FSOM is exploited in gas agent detection [24] and human vigilance analysis system [58]. In this thesis, the FSOM was also proved useful in pattern recognition-based fault diagnosis. The FSOM was also successfully used in identification of the laboratory-scaled process. Especially, the FSOM using Sugeno's rules produce excellent results compared to the original FSOM. The FSOM was also used as a state controller. The model-based learning scheme, where the controller is tuned based on a model of the process, was used.

There are many open questions in the neural and neuro-fuzzy control systems. It is obvious that the benefits of neuro-fuzzy control appear in more complicated control schemes and tasks. Adaptive and multiple-input multiple-output control systems are very interesting application areas. The research field is still quite young, but it is getting more important.

All together, the FSOM was found as a potential neuro-fuzzy system introducing fresh ideas, which need still further development. It is also sensible to incorporate ideas of the other neuro-fuzzy systems into more sophisticated versions of the FSOM.

References

- [1] C. W. Anderson, "Learning to control an inverted pendulum using neural networks," *IEEE Contr. Syst. Mag.*, pp. 31–36, Apr. 1989.
- [2] H. R. Berenji, "Fuzzy logic Controllers," in *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, R. R. Yager and L.A. Zadeh, Eds., Kluwer academic publishers, 1992, pp. 69–96.
- [3] H. Berenji, "A reinforcement learning-based architecture for fuzzy logic control," *Int. J. Approximate Reasoning*, no. 6, pp. 267–292, 1992.
- [4] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 724–740, Sept. 1993.
- [5] J. C. Bezdek, E. C.-K. Tsao, and N. R. Pal, "Fuzzy Kohonen clustering networks," in *Proc. 1st IEEE Int. Conf. Fuzzy Systems, Fuzz-IEEE'92*, 1992, pp. 1035–1043.
- [6] S. A. Billings and W. S. F. Voon, "Correlation based model validity tests for nonlinear model," *Int. J. Control*, vol. 44, no. 1, pp. 235–244, 1986.
- [7] G. A. Carpenter, S. Grossberg, and D. B. Rosen, "Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system," *Neural Networks*, vol. 4, pp. 759–771, 1991.
- [8] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 698–713, 1993.
- [9] S. Chen, F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Networks*, vol. 2, No. 2, pp. 302–309, 1991.
- [10] F.-C. Chen, "Back-propagation neural networks for nonlinear self-tuning adaptive control," *IEEE Control Systems*, pp. 44–48, 1990.
- [11] A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, John Wiley & Sons Ltd., 1992.

-
- [12] F. L. Chung and T. Lee, "Fuzzy competitive learning," *Neural Networks*, vol. 7, no. 3, pp. 539–551, 1994.
- [13] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–192, 1989.
- [14] W. F. Ganong, *Review of Medical Physiology*, Prentice-Hall International, Inc., 1987.
- [15] M. M. Gupta and D. H. Rao, "On the principles of fuzzy neural networks," *Fuzzy Sets and Systems*, vol. 61, pp. 1–18, 1994.
- [16] S. K. Halgamuge and M. Glesner, "Neural networks in designing fuzzy systems for real world applications," to appear in *Int. J. Fuzzy Sets and Systems*, 1994.
- [17] S. K. Halgamuge, W. Poechmueller, and M. Glesner, "An alternative approach for generation of membership functions and fuzzy rules based on radial and cubic basis function networks," Darmstadt Univ. of Techn., Inst. of Microelectronic Systems, Tech. Rep., 1994.
- [18] S. Horikawa, T. Furahashi, and Y. Uchikawa, "On fuzzy modeling using fuzzy neural networks with the backpropagation algorithm," *IEEE Trans. Neural Networks*, vol. 3, pp. 801–806, 1992.
- [19] K. J. Hunt and D. Sbarbaro, "Neural networks for nonlinear internal model control," *IEE Proc.-D*, vol. 138, no. 5, pp. 431–438, 1991.
- [20] T. L. Huntsberger and P. Ajjimarangsee, "Parallel self-organizing feature maps for unsupervised pattern recognition," *Int. J. General Systems*, vol. 16, no. 4, pp. 357–372, 1990.
- [21] J.-S. R. Jang, "Self-learning fuzzy controllers based on temporal back propagation," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 714–723, 1992.
- [22] J.-S. R. Jang, "ANFIS: adaptive-network-based fuzzy inference system," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, no. 3, pp. 665–685, 1993.
- [23] J.-S. R. Jang, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Trans. Neural Networks*, vol. 4, no.1, pp. 156–159, 1993.
- [24] T. Jukarainen and P. Vuorimaa, "Gas recognition using fuzzy self-organizing map," in *Proc. 1st TUT Symposium Signal Processing*, Tampere, Finland, May 20, 1994, pp. 36–38.
- [25] J. A. Kangas, T. K. Kohonen, and J. T. Laaksonen, "Variants of self-organizing maps," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 93–99, 1990.

- [26] P. Kanerva, *Sparse Distributed Memory*, MIT Press, London, 1988.
- [27] T. Kohonen, *Self-Organization and Associative Memory*, 2nd ed. Berlin: Springer Verlag, 1988.
- [28] T. Kohonen, "Self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.
- [29] S.-G. Kong and B. Kosko, "Adaptive fuzzy systems for backing up a truck-and-trailer." *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 211–223, 1992.
- [30] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Approach to Machine Intelligence*, Englewood Cliffs, NJ: Prentice Hall, 1992.
- [31] B. Kosko, "Stochastic competitive learning," *IEEE Trans. Neural Networks*, vol. 2, no. 5, pp. 522–529, 1991.
- [32] Y.-H. Kuo, C.-I. Kao, and J.-J. Chen, "A fuzzy neural network model and its hardware implementation," *IEEE Trans. Fuzzy Systems*, vol. 1, no. 3, pp. 171–183, 1993.
- [33] T. M. Kwon and M. E. Zervakis, "A self-organizing KNN fuzzy controller and its neural networks structure," *Int. J. Adaptive Control and Signal Processing*, vol. 8, pp. 407–431, 1994.
- [34] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller – part I," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 404–418, 1990.
- [35] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller – part II," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 419–435, 1990.
- [36] S. Lee and R. M. Kil, "A gaussian potential function network with hierarchically self-organizing learning," *Neural Networks*, vol. 4, pp. 207–224, 1991.
- [37] S. C. Lee and E. T. Lee, "Fuzzy neural networks," *Mathematical Biosciences*, vol. 23, pp. 151–177, 1975.
- [38] C.-T. Lin and C. S. G. Lee, "Neural-network-based fuzzy logic control and decision system," *IEEE Trans. Computers*, vol. 40, no. 12, pp. 1320–1336, 1991.
- [39] C.-T. Lin and C. S. G. Lee, "Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems," *IEEE Trans. Fuzzy Systems*, vol. 2, no. 1, pp. 46–63, 1994.
- [40] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, vol. 4, no. 2, pp. 4–22, 1987.

- [41] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computing*, 1, pp. 281–294, 1989.
- [42] M. T. Musavi, M. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, "On the training of radial basis function classifiers," *Neural networks*, vol. 5, pp. 595–603, 1992.
- [43] D. Nauck and R. Kruse, "A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backpropagation," in *Proc. IEEE Int. Conf. Neural Networks ICNN'93*, 1993, pp. 1022–1027.
- [44] D. Nauck, F. Klawonn, and R. Kruse, "Combining neural networks and fuzzy controllers," *FLAI'93*, Linz, Austria, Jun. 28–Jul. 2, 1993.
- [45] D. H. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Control Systems Mag.*, pp. 18–23, 1990.
- [46] J. Nie and D. A. Linkens, "Learning control using fuzzified self-organizing radial basis function network," *IEEE Trans. Fuzzy Systems*, vol. 1, no. 4, pp. 280–287, 1993.
- [47] E. Oja, "Neural networks, principal components and subspaces," *Int. J. Neural Systems*, vol. 1, no. 1, pp. 61–68, 1989.
- [48] T. Ojala and P. Vuorimaa, "Fuzzy self-organizing map using Sugeno's fuzzy rules," in *Proc. 1st TUT Symposium Signal Processing*, Tampere, Finland, May 20, 1994, pp. 169–172.
- [49] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Trans. Neural Networks*, vol. 3, pp. 683–697, 1992.
- [50] W. Pedrycz, "Fuzzy neural networks with reference neurons as pattern classifiers," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 770–775, 1992.
- [51] F. Poirier and A. Ferrieux, "DVQ: Dynamic vector quantization— an incremental LVQ," in *Artificial Neural Networks*, T. Kohonen, K. Mäkisara, O. Simula and J. Kangas, Eds., Elsevier Science Publishers B.V. (North-Holland), 1991.
- [52] V. Ruoppila, *Takaisinkytketty neuraaliverkkosäätäjä*, Diplomityö, Tampere, TTKK, sähkötekniikan osasto, 1992.
- [53] J. Suontausta and V. T. Ruoppila, and H. N. Koivo, "Modelling of non-linear systems using radial basis function networks," in *Proc. 10th IFAC Symposium on System Identification, SYSID '94*, Copenhagen, Denmark, July 4–6, 1994.
- [54] T. Sorsa, *Neural Network Approach to Fault Diagnosis*, Licentiate thesis, Tampere, TTKK, Department of Electrical Eng., 1993.

- [55] C.-T. Sun, "Rule-base structure identification in an adaptive-network-based fuzzy inference system," *IEEE Trans. Fuzzy Systems*, vol. 2, no. 1, pp. 64–73, 1994.
- [56] H. Takagi, N. Suzuki, T. Koda, and Y. Kojima, "Neural networks designed on approximate reasoning architecture and their applications," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 752–760, 1992.
- [57] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 1, pp. 116–132, 1985.
- [58] M. Tervonen, P. Vuorimaa, A. Värri, and J. Hasan "A human vigilance analysis system using neural networks and fuzzy logic," in *Proc. 1st TUT Symposium Signal Processing*, Tampere, Finland, May 20, 1994, pp. 39–42.
- [59] S. Thaler, "Fuzzy rule generation based on a neural network approach," *Electronic Engineer*, pp. 43–50, July, 1993.
- [60] E. C.-K. Tsao, J. C. Bezdek, and N. R. Pal, "Fuzzy Kohonen clustering networks," *Pattern recognition*, vol. 27, no. 5, pp. 757–764, 1994.
- [61] P. Vuorimaa, "Fuzzy self-organizing map," *Fuzzy Sets and Systems*, vol. 66, pp. 223–231, 1994.
- [62] P. Vuorimaa, "Use of the default rule in fuzzy self-organizing map," to appear in *Advances in Fuzzy Theory & Technology*, vol. II, 1994.
- [63] P. Vuorimaa, "Self-generating fuzzy self-organizing map," submitted to *IEEE Trans. Fuzzy Systems*.
- [64] P. Vuorimaa, "Use of the fuzzy self-organizing map in pattern recognition," in *Proc. 3rd IEEE Int. Conf. Fuzzy Systems, FUZZ-IEEE'94*, Orlando, USA, June 26-29, 1994, pp. 798–801.
- [65] P. Vuorimaa, "A model-based neuro-fuzzy controller," in *Proc. Conf. Artificial Intelligence Research in Finland, STeP-94*, Turku, Finland, Aug. 29–31, 1994, pp. 177–183.
- [66] P. Vuorimaa and T. Jukarainen, "A neuro-fuzzy system for chemical agent detection," submitted to *IEEE Trans. Fuzzy Systems*, 1994.
- [67] L.-X. Wang and J. M. Mendel, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 807–814, 1992.

-
- [68] R. R. Yager, D. P. Filev, and T. Sadeghi, "Analysis of flexible structured fuzzy logic controllers," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 7, pp. 1035–1043, July 1990.
- [69] K. J. Åström and B. Wittenmark, *Computer Controlled Systems: Theory and Design*, Prentice-Hall, inc., 1990.