Data Analysis with Neuro-Fuzzy Methods

Habilitationsschrift

zur Erlangung des akademischen Grades

doctor rerum naturalium habilitatus

(Dr.rer.nat.habil.)

der Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg,

vorgelegt von Dr. rer. nat. Detlef D. NAUCK

Magdeburg, 25. Februar 2000

Data Analysis with Neuro-Fuzzy Methods

Habilitationsschrift

zur Erlangung des akademischen Grades

doctor rerum naturalium habilitatus

(Dr.rer.nat.habil.),

genehmigt
durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dr. rer. nat. Detlef D. NAUCK

geb. am 20. März 1964
in Braunschweig

Gutachter:

Prof. Dr. Rudolf Kruse
Prof. Dr. Adolf Grauel
Prof. Dr. Raul Rojas

Magdeburg, 8. März 2000

Contents

Abstract

In this thesis neuro-fuzzy methods for data analysis are discussed. We consider data analysis as a process that is exploratory to some extent. If a fuzzy model is to be created in a data analysis process it is important to have learning algorithms available that support this exploratory nature. This thesis systematically presents such learning algorithms, which can be used to create fuzzy systems from data. The algorithms are especially designed for their capability to produce interpretable fuzzy systems. It is important that during learning the main advantages of a fuzzy system – its simplicity and interpretability – do not get lost. The algorithms are presented in such a way that they can readily be used for implementations. As an example for neuro-fuzzy data analysis the classification system NEFCLASS is discussed.

Kurzfassung

In dieser Arbeit werden Neuro-Fuzzy-Methoden zur Datenanalyse untersucht. Wir betrachten Datenanalyse als einen im gewissen Umfang explorativen Prozeß. Wenn ein Fuzzy-Modell im Rahmen eines solchen Prozesses erzeugt werden soll, sind Lernverfahren von hoher Bedeutung, die diesen explorativen Ansatz unterstützen können. Die Arbeit erarbeitet systematisch eine Reihe von Lernverfahren, die in der Lage sind, Fuzzy-Systeme aus Daten zu erzeugen. Diese Algorithmen werden insbesondere so konstruiert, daß sie interpretierbare Fuzzy-Systeme generieren können. Es ist wichtig, daß die Hauptvorteile eines Fuzzy-Systems – seine Einfachheit und Interpretierbarkeit – im Laufe eines Trainingsvorgangs nicht verloren gehen. Die Algorithmen werden in einer Form angegeben, die eine direkte Anwendbarkeit in Implementierungen ermöglicht. Als ein Beispiel für Neuro-Fuzzy-Datenanalyse diskutieren wir das Klassifikationssystem NEFCLASS.

Chapter 1

Introduction

One of the most challenging tasks in computer science is to build machines or computer programs that are capable of learning. We expect a learning machine or program to improve automatically with experience gathered during its life cycle. One of the first learning machines that was ever built is the perceptron by Frank Rosenblatt. The perceptron was built to imitate some functions of an eye. It had an artificial retina that could detect light. From a pattern of bright and dark points it decided which class the presented pattern belongs to. The exciting feature of the perceptron was that it was able to learn to classify patterns correctly after it had "seen" a few examples.

The perceptron was an implementation of an early artifical neural network model [Rosenblatt, 1958, Rosenblatt, 1962]. Artifical neural networks – or neural networks, for short – were created to simulate some aspects of the learning capabilities of human brains. A neural network consists of a number of simple processing elements called units or artificial neurons. Artifical neurons were introduced by McCulloch and Pitts in 1943 and are used to simulate the behaviour of nerve cells. A nerve cell collects input signals and integrates them over time. If a certain threshold is reached the cell becomes active and emits a signal with high frequency. Artificial neurons do not usually simulate the temporal aspects and just sum up incoming signals and denote their activity by some output value if the input is larger than some threshold [McCulloch and Pitts, 1943].

Artifical neurons exchange signals along weighted connections (network structure) that are used to simulate synapses. The signals are changed when they travel along the connections: They are combined – usually by multiplication – with the connection weights. A neuron gathers the input from all incoming connections to compute its activation value. In 1949 Hebb hypothesized about learning in networks of artifical neurons. The Hebb rule encodes the correlations of activations of connected units in the weights. A weight is increased, if the two neurons that are connected by it are active at the same time. The weight is decreased, if only one of the two

connected neurons is active [Hebb, 1949]. The perceptron used a variation of this rule.

The units of most types of neural networks are organized in layers which are called input layer, hidden layer(s) or output layer, depending on their functionality. A neural network with n units in its input layer and m units in its output layer, implements a mapping $f : X^n \longrightarrow Y^m$. The sets $X^n \subseteq \mathbb{R}^n$ and $Y^m \subseteq \mathbb{R}^m$ are the input and output domains. Hidden layers add to the complexity of a neural network, and are important, if arbitrary mappings have to be represented.

Soon after the presentation of the perceptron the importance of hidden layers became apparent. Minsky &Papert showed that a perceptron – which only has an input and an output layer – is not able to solve linearly non-separable tasks [Minsky and Papert, 1969]. It was known that this drawback could be resolved by introducing a hidden layer, but a learning algorithm was not available before Rumelhart, Hinton & Williams, in 1986, rediscovered a learning rule that was first published by Werbos in 1974 and which became known as *backpropagation* [Rumelhart et al., 1986a, Werbos, 1974]. Since then neural networks have become very popular and their learning capabilities make them very interesting not only for research but also for industrial applications.

Nowadays neural networks are important tools used for classification, process automation, time series prediction, data mining, etc. We know that certain types of neural networks are universal approximators [Hornik et al., 1989, Poggio and Girosi, 1989]. A comprehensive introduction to neural networks can be found, for example, in [Haykin, 1994, Rojas, 1996, Zurada, 1992]. Neural networks are a part of machine learning, an area of practical computer science where phenomena of learning are studied [Mitchell, 1997]. Machine learning is an interdisciplinary field with connections to artifical intelligence, information theory, statistics, cognitive science but also fields like neurophysiology, psychology and biology.

Machine learning does not only consider algorithms that enable machines or programs to learn, but also the way the knowledge obtained in this process is represented. Ideally a learning system is able to tell its user what it has learned. Thus a user not only learns about the problem of interest, he or she can also check whether the knowledge represented within the learning system is correct or at least plausible.

A good example are decision trees that recursively partition a hypothesis space and represent the result in the form of a tree. The decision tree cannot only be used for prediction, but also reveals how the system learns to partition the hypothesis space and how a prediction is computed. The tree can also be transformed into a set of rules. The induction algorithm that creates the decision tree has a preference for shorter trees. Thus the system is not only transparent but also compact and so easier to comprehend.

The most often used approach to learning systems is induction. This means a

learning algorithm processes examples that provide the correct answer or output for a given input. When the examples for a learning algorithm consist of real world data then they are usually tainted with noise, ambiguity, uncertainty, imprecision, vagueness or incompleteness. Classical models try to avoid such phenomena because they are considered to have a negative influence on learning or inference processes. But it is possible to deliberately make use of this kind of information.

In 1965 Zadeh suggested fuzzy sets, generalizations of regular crisp sets [Zadeh, 1965]. A fuzzy set can contain elements with different degrees of membership between 0 and 1. Fuzzy sets can be conveniently used to represent linguistic expressions like "the temperature is *low*", "this person is *tall*", etc. Expressions like this are common in our everyday life and we can easily handle them and use them to draw conclusions. If-then rules that use linguistic expressions (e.g. "if x is *small* then y is *approximately zero*") are called linguistic rules or fuzzy rules. Systems based on fuzzy rules are called fuzzy systems.

Fuzzy systems became popular in the area of control engineering in the form of fuzzy controllers. For this and other kinds of applications of fuzzy systems the term "fuzzy logic" is widely used. However, most applications of fuzzy systems do not use generalized logical rules. Fuzzy systems as they are considered in this thesis use fuzzy sets to represent gradual qualities. A fuzzy rule is then a vague description of a sample of some partially known function. Such fuzzy systems belong to "fuzzy logic in the broad sense". In contrast "Fuzzy logic in the narrow sense" refers to systems that use logical calculi and deduction mechanisms to extend classical two-valued logic to the unit interval as the set of truth values. In this thesis only "fuzzy logic in the broad sense" is considered.

By partitioning variables with overlapping fuzzy sets instead of crisp sets, fuzzy systems can model smooth transitions between states and thus avoid counterintuitive results that can occur when boundaries of crisp states are crossed. Fuzzy sets let us observe a space of hypothesis under a certain granularity such that we do not need to distinguish between very similar values. This results in an information compression and helps in building simple inexpensive solutions that can be intuitively understood.

Zadeh coined the term *soft computing* to denote approaches to human reasoning that deliberately make use for human tolerance of uncertainty and vagueness to obtain inexpensive solutions that are easy to handle [Zadeh, 1994a]. In addition to fuzzy systems, neural networks, evolutionary computation and probabilistic reasoning and their combinations are considered as soft computing. Because these areas are all based on numeric methods they are also subsumed under the notion *computational intelligence* – a term suggested by Bezdek to characterize numeric approaches to modelling intelligent behaviour in contrast to artificial intelligence, where mostly methods based on symbol manipulation are considered [Bezdek, 1994].

Nowadays fuzzy systems are – in addition to control application – applied in data analysis problems like classification, function approximation or time series predic-

tion. Their advantage is that they can provide simple intuitive models for interpretation and prediction. Prior knowledge in the form of fuzzy rules can be easily integrated. In order to use fuzzy systems in data analysis it must be possible to learn them from examples. Learning in fuzzy systems was first used for fine-tuning fuzzy controllers, where reinforcement learning strategies were applied [Berenji, 1990, Nauck, 1994b, Shao, 1988]. These approaches are not considered in this thesis, because they are not applicable to data analysis, where supervised learning methods are needed, i.e. example-based (inductive) learning.

The applicaton of fuzzy systems to data analysis is also known as "fuzzy data analysis". This term refers to the analysis of crisp data with fuzzy methods. There are also approaches that consider the analysis of fuzzy data with generalized statistics [Kruse and Meyer, 1987]. Such approaches are not considered in this thesis.

Learning in fuzzy systems is most often implemented by learning techniques derived from neural networks. The term *neuro-fuzzy system* (also neuro-fuzzy methods or models) refers to combinations of neural networks and fuzzy systems. This combination does not usually mean that a neural network and a fuzzy system are used together in some way. A neuro-fuzzy method is rather a way to create a fuzzy system from data by some kind of (heuristic) learning method that is motivated by learning procedures used in neural networks.

Equivalent terms for *neuro-fuzzy* that can be found in the literature are *neural fuzzy* or sometimes *fuzzy networks*. We distinguish these terms from *fuzzy neural networks* or *fuzzy-neuro approaches* which denote fuzzified neural networks, i.e. neural networks that use fuzzy sets instead of real numbers as weights, activations, inputs and outputs [Buckley and Eslami, 1996, Feuring and Lippe, 1996]. Fuzzy neural networks are black box function approximators that map fuzzy inputs to fuzzy outputs. They are obtained by applying Zadeh's extension principle to a normal neural network architecture. We do not discuss these approaches further.

Learning in fuzzy systems must consider structure learning, i.e. creation of a rule base, and parameter learning, i.e. optimization of fuzzy sets. Parameter learning is often done by algorithms that were inspired by neural network learning. Structure learning on the other hand is usually not taken from neural networks. In neural networks only few approaches exist to automatically determine the number of hidden nodes. Cascade correlation is such an approach [Fahlman and Lebiere, 1990], but it is not appropriate for learning fuzzy rule bases. Usually the structure of a neural network is given and only the parameters (connection weights) are trained. The term "neuro-fuzzy", however, is nowadays applied to almost all approaches to learning in fuzzy systems such that the learning of fuzzy rules is also subsumed under this notion [Jang et al., 1997, Nauck et al., 1997].

Distinctions are only made, for example, when fuzzy rules are created by fuzzy decision tree learning [Janikow, 1998] or by genetic algorithms [Hopf and Klawonn, 1994, Kinzel et al., 1994, Lee and Takagi, 1993, Takagi and

Lee, 1993]. There are also approaches that try out combinations of variables, if the dimensionality of the considered problem is low [Krone and Kiendl, 1996].

In this thesis neuro-fuzzy methods for data analysis are discussed. We consider data analysis as a process that is exploratory to some extent, as in intelligent data analysis [Berthold and Hand, 1999] or data mining [Fayyad et al., 1996]. If a fuzzy model is to be created in such a scenario it is important to have learning algorithms available that support the exploratory nature of the data analysis process. This thesis systematically presents such learning algorithms, which can be used to create fuzzy systems from data. The algorithms are especially examined for their capabilities to produce interpretable fuzzy systems. It is important that during learning the main advantages of a fuzzy system – its simplicity and interpretability – do not get lost. The algorithms are presented in a way that they can readily be used for implementations. As an example for neuro-fuzzy data analysis a classification system (NEFCLASS) is discussed.

The following chapter introduces intelligent data analysis and knowledge discovery in databases as areas where neuro-fuzzy systems can be applied with great benefits. We characterize data analysis as an approach with an exploratory nature and that learning algorithms used to create neuro-fuzzy models must support this process.

In Chapter 3 foundations of fuzzy systems, neural networks and neuro-fuzzy systems are reviewed to introduce the necessary notations used throughout the following chapters. One section of the chapter is devoted to the discussion of the characteristics of interpretable fuzzy systems.

Chapters 4 and 5 are the main parts of the thesis. Here algorithms for learning fuzzy rules and for optimizing membership functions are presented. For fuzzy rule learning methods from cluster analysis, fuzzy decision tree learning and approaches specially developed for neuro-fuzzy systems are discussed. For optimizing fuzzy sets, methods based on gradient descent and special heuristics for neuro-fuzzy systems are analysed. We also discuss problems caused by learning techniques based on rule weights, which are often used, because they are so easy to implement. In addition methods for handling data analysis problems where the data has symbolic and numeric values are presented and ways for learning fuzzy rules under missing values are shown.

We refrain from discussing several neuro-fuzzy approaches that usually only differ in the way they represent fuzzy systems. Instead we follow a more general approach and organize the chapters according to the kind of learning algorithms and the kind of fuzzy systems that can be created by them. We do not consider learning algorithms that are not applicable in data analysis like, for example, reinforcement learning. Different kinds of neuro-fuzzy approaches can be simply obtained by suitably combining the learning algorithms presented. For an overview on different neuro-fuzzy approaches see, for example, [Jang et al., 1997, Nauck et al., 1997].

Chapter 6 presents the NEFCLASS system as an example for constructing a neuro-fuzzy system for data analysis in the context of classification. We provide some guidelines for implementation and discuss some applications of NEFCLASS. The thesis is concluded in Chapter 7.

Chapter 2

Intelligent Data Analysis

Data analysis can be described as the process of computing summaries and derived values from data, or – more general – as the process of converting data into information. In this thesis data is understood as a collection of values or recordings – numeric or otherwise – that describe the attributes of objects under study. Information or knowledge that is generated from data is used here in a very general meaning. It may be, for example, a set of induced rules used to draw inferences, a linguistic description to summarize or simplify the nature of a data set, or a more or less complex model to accurately describe the processes that underlie observed data. We consider information as a necessary prerequisite for decision making. We assume that data is on a much lower level than information such that it cannot be readily put to use. Data sets may be, for example, very large, high-dimensional, ridden with noise, conflicting, incomplete, vague or uncertain. Thus without proper processing and analysis, raw data itself is usually quite useless for decision making.

Our modern world is data-driven. Many decisions are made based on the analysis of data. Examples of typical application areas are the weather forecast, stock prediction, the identification of prospective customers, object recognition in images, etc. The remarkable progress in computer technology not only allows us to gather and store more data than we can possibly analyse, it also enables us to do analyses one could not think of 30 years ago. Thus not only the need for data analysis has increased, but also the number of feasible data analysis methods [Berthold and Hand, 1999].

This chapter discusses modern approaches to data analysis like *intelligent data analysis* and *knowledge discovery in databases* (KDD) and *data mining* to set the stage for neuro-fuzzy methods in data analysis which are discussed in this thesis. Section 2.1 considers the term "intelligent data analysis" and Section 2.2 focuses on KDD and data mining.

2.1 What is Intelligent Data Analysis?

The notion of "intelligent data analysis" is used to describe a certain approach to data analysis. Like many notions that feature the term "intelligent" intelligent data analysis also has no exact definition. The discussion in this section follows the line of argumentation by Hand found in [Hand, 1998] and in Chapter 1 of [Berthold and Hand, 1999].

Data analysis always has some objective, i.e. by analysing data we hope to answer questions. We can distinguish between

- exploratory data analysis, and
- confirmatory data analysis.

The exploratory approach seeks to answer questions like: Are there relevant structures in the data? Are there anomalous records? How can the data conveniently be summarized? The confirmatory approach is interested in questions like: Are these two groups different? Can the value of this attribute be predicted from measured values?

We can also distinguish between

- descriptive data analysis, and
- inferential data analysis.

In descriptive analysis we are interested in statements about the considered data set, for example, proportions of certain values or characteristic values, like: How many children were born in Germany in 1998? Inferential analysis is often based on samples from some population and aims at drawing conclusions, e.g. what can we say about the number of births in Germany in the following year?

Hand points out that data analysis must be understood as a process [Berthold and Hand, 1999]. It is not a case of simply selecting and applying a data analysis method or tool to a particular problem to obtain the desired answer (cookbook fallacy). Data analysis is not a collection of independent tools. The available methods have rather complex interrelationships and there is no perfect tool for a problem of interest. Several methods are usually applicable, each with subtle differences, preconditions or assumptions. Furthermore, research questions to be answered by data analysis are usually not formulated precisely enough to justify the application of one method alone. Very often the analysis reveals new questions that must be answered by applying further methods, or the initial question changes during analysis and this process can be iterated several times.

One aspect of intelligent data analysis is therefore the repeated application of methods and the ongoing refinement of the questions to be answered by the analysis in a carefully planned and considered manner.

The two most important areas that contribute to intelligent data analysis are statistics and machine learning. Statistics has its roots in mathematics and is driven by the notion of a model – a postulated structure, or an approximation to a structure, which could have led to the data [Berthold and Hand, 1999]. Machine learning is an area of practical computer science. In [Mitchell, 1997] we read: "machine learning involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge of the learner". So both areas are concerned with quite similar questions, where statistics used to focus on mathematical rigour and machine learning is driven by the goal of building learning systems with the help of the computer. There is a great potential for synergy between both fields as, for example, artifical neural network technology shows.

Certainly further areas that also contribute to intelligent data analysis can be identified. For example, soft computing techniques [Zadeh, 1994a] like fuzzy systems, neural networks and probabilistic reasoning – areas that deliberately exploit the tolerance for uncertainty and vagueness in the area of cognitive reasoning. This can lead to considerable reduction in complexity when real-world problems have to be solved and can lead to solutions that are easy to handle, robust, and low-priced. Other areas that can be mentioned are KDD, evolutionary computation, artificial intelligence or approximation theory and also database theory that provides means to handle large amounts of data.

The term "model" is widely used in data analysis, but with different interpretations [Berthold and Hand, 1999]. It is possible to distinguish between empirical and mechanistic models. Empirical models describe relationships without basing them on an underlying theory, while mechanistic models describe an underlying reality that is responsible for the observed relationships. We can also distinguish between models for prediction and models for understanding. In addition there is the distinction between model and pattern. Here "pattern" is not used in the sense of a point, a case, or a vector as it is used in pattern recognition but in the sense of a local structure. Data mining is often concerned with the detection of patterns in data (Section 2.2). A model on the contrary is a global structure summarizing relationships over many cases. However, in data analysis the emphasis is not on modelling but on answering questions. "It is these questions, not the model per se, which must be paramount" [Hand, 1998].

The advances in computer technology allow us to quickly fit a large number of models to large data sets – an approach that was not possible 30 years ago. Data analysts are nowadays not forced to do long theoretical studies in order to hopefully select an appropriate method that is well suited to their data and then to use up all available computer time in one attempt to estimate the parameters of the selected model and

to live with the consequences. The possibility of trying out several methods may sometimes lead to less formal and less mathematically justified approaches in data analysis, but it has also made data analysis available for a wide range of applications.

We cannot expect that an analyst is an expert in the analysis methods he applies for a problem from his domain. In the same way we do not expect a driver to be capable of repairing his car or a computer user to understand the function of an arithmetic and logic unit. Geologists or physicians are not interested in the mathematical foundations of the analysis methods they apply to their data but in the answers to questions like where to drill for oil, or which treatment is best for a certain disease. Of course, some foundational understanding about the characteristics and prerequisites of the applied methods must be available, otherwise one cannot expect to obtain useful results and the danger of misuse is great. But the point is that data analysis is a practical area and data analysis methods nowadays – with the help of the computer – are used as tools.

From a practical point of view certain restrictions have to be imposed on models obtained in a data analysis process. Again thanks to the computer it is possible to create almost arbitrarily sophisticated models that fit every subtle aspect of a data set or an underlying process. Not only are such subtleties usually irrelevant in practical applications, complex models also tend to overfit the data, i.e. they fit the noise and uncertainties contained in the data. From the viewpoint of a user a model must also be comprehensible, interpretable and inexpensive. In several application areas, e.g. medicine or financial services, reasons of security demand that models can only be trusted, if they can be understood by the user. For example an artificial neural network that was created from medical data will probably not be simply accepted as a decision authority, if it recommends an amputation based on the data of a patient. Models obtained from data analysis that are applied in practice usually require transparency and interpretability in terms of the attributes they process. This also requires small models because models with many parameters are not comprehensible to a user.

Therefore another aspect of intelligent data analysis is to select an appropriate model with the application in mind. It may be necessary to sacrifice precision for interpretability, i.e. a suitable balance between model complexity and comprehensibility, between precision and simplicity must be found.

A third aspect of intelligent data analysis requires the selection of appropriate algorithms for the process of creating a model. There can be several algorithms available for creating the same kind of model and they may not only differ in computational complexity, speed of convergence, ease of parameterization, but also in the way they ensure certain features in the model they create from data. For example, there are many approaches to learning fuzzy systems from data. In Chapters 4 and 5 we will especially focus on simple algorithms that allow us to control the interpretability of the created fuzzy model.

Finally we can say that the process of data selection is important for intelligent data analysis. This requires that data be collected and structured with the analysis in mind. Unfortunately, this is very often not the case, for example, in industry. Here data is often collected randomly before the need for an analysis arises. Therefore data preprocessing, for example, the detection of outliers and anomalies, the analysis of the pattern of missing values, etc. is an important task.

To summarize, we can say that intelligent data analysis is a process of critical assessment, exploration, testing and evaluation. It requires the application of knowledge and expertise about the data and it is fundamentally interdisciplinary [Berthold and Hand, 1999].

In this thesis neuro-fuzzy methods for data analysis are discussed. They are regarded as a valuable method for intelligent data analysis as they are especially useful, if simple and interpretable models are required. There exist algorithms that can quickly create models and thus support the exploratory approach of intelligent data analysis. Neuro-fuzzy approaches can be seen as a technique to create models that act as a bridge between models for understanding and models for prediction. Neuro-fuzzy methods, as will be described in Section 3.3, create fuzzy models from data that can on the one hand be used for prediction, but on the other hand are also interpretable in an intuitive linguistic way and are therefore useful to describe the underlying data. These aspects are also discussed in Section 3.4.

2.2 Knowledge Discovery in Databases and Data Mining

As already described in the last section the advances in computer technology enable us to store vast amounts of data. For example a modern supermarket stores each transaction of every customer. For large chains such data can quickly amount to gigabytes of data each day. The analysis of such large databases requires special data analysis methods. Not only the amount of data poses a problem, but also the goal of the analysis. Very often it is not clear what kind of information or knowledge can be obtained by data analysis. Note that we use the terms “information” and “knowledge” interchangeably in the sense described in Section 2.1.

Knowledge discovery in databases (KDD) is a research area that considers the analysis of large data bases in order to identify valid, useful, meaningful, unknown, and unexpected relationships [Fayyad et al., 1996]. This definition of KDD shows that it is an exploratory process. In the beginning we might not know what kind of knowledge or information we are looking for. Techniques for detecting patterns, i.e. local structures, anomalies or dependencies are required. Like intelligent data analysis (Section 2.1), KDD does not place an emphasis on modelling, but on answering questions. However, one could say that KDD goes one step further, because

it "tries to answer question we did not ask" in the beginning of the KDD process, that means KDD tries to discover unknown and unexpected relationships.

Like intelligent data analysis, KDD is a highly interdisciplinary area and interacts with statistics, machine learning, soft computing, artificial intelligence and data base theory.

One of the techniques applied in KDD is called *data mining*. The term "data mining" is meant to illustrate the exploratory nature of the analysis process and it is used to describe the application of different machine learning methods and data analysis techniques in order to search for knowledge in data. This means, data mining can be seen as a tool in a KDD process to automatically obtain prognostic or explanatory information from large data collections.

Data mining cannot be understood as a single method. There are many different techniques that are used in data mining, for example, statistics, machine learning, probabilistic networks, neural networks, fuzzy systems and combinations of these, like neuro-fuzzy systems.

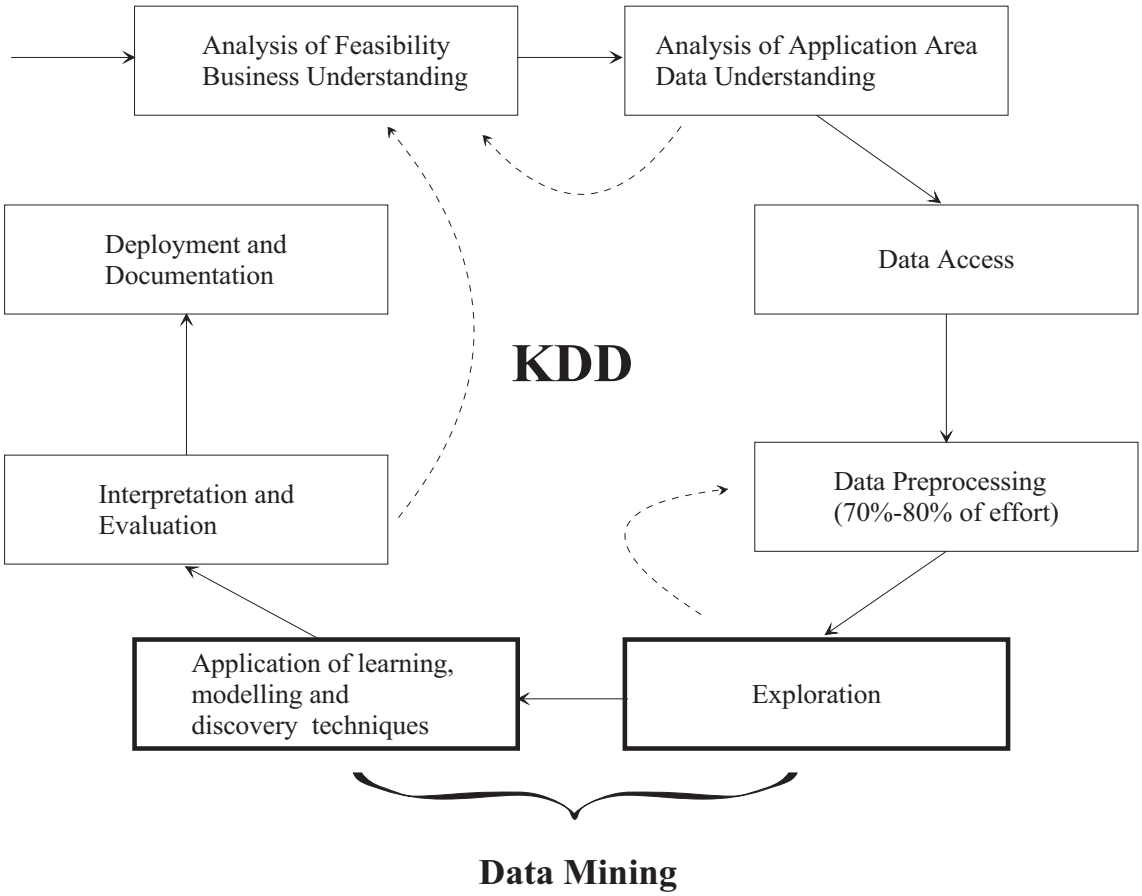


Figure 2.1: A process model for KDD

Figure 2.1 shows a process model of KDD that combines the process models used in [Fayyad et al., 1996, Nakhaeizadeh, 1998] and the European research project CRISP-DM (CRoss Industry Standard Process for Data Mining, see http://www.ncr.dk/crisp). The two points, *exploration* and *application of learning, modelling and discovery techniques* are together called data mining. KDD and data mining are usually very application-oriented. This is obviously due to the fact that most data is gathered in industry. Of course, KDD is also relevant in scientific areas, for example, detecting patterns in data from weather satellites to predict hurricanes, analysis of genetic data etc.

Data preprocessing is usually the most expensive part of KDD. This is due to the fact that real world data is often faulty. Many cases and attributes may have missing values. These cases cannot be simply left out from analysis, but the pattern of missing values must be analysed, especially if their percentage is high. Questions like "what kind of values are missing and why" must be answered. Real world data is also very often high-dimensional and there are a large number of cases ("large" means at least a few ten thousands of cases). It is often not possible to analyse such data sets completely, so that subsets of useful attributes and cases must be selected.

Data warehousing [Inmon, 1996] is an area of data base theory that aims at providing data relevant for decision making. A data warehouse is a data base that is selected from one or several operative data bases. It contains all data that is relevant for the business processes of a company and processes and aggregates these data appropriately. Thus in a KDD process a data warehouse provides the necessary data access facilities and part of the preprocessing in order to execute data mining applications.

The goal of data mining is to answer questions. The following areas are especially important in data mining [Fayyad et al., 1996, Nakhaeizadeh, 1998]:

- segmentation (e.g. what kind of customers does a company have?)
- classification (e.g. is this person a prospective customer?)
- concept description (e.g. what attributes describe a prospective customer?)
- prediction (e.g. what value will the stock index have tomorrow?)
- deviation analysis (e.g. why has the behaviour of customers changed?)
- dependency analysis (e.g. how does marketing influence customer behaviour?)

Data mining is comparable to intelligent data analysis with especially strong emphasis on the exploratory nature of the analysis process. The questions to be answered often arise only after the data mining process begins. For instance, exploration strategies like graphical visualization are important tools in data mining, in order to gain some insight into the data and to guide further analysis.

As in intelligent data analysis the focus in data mining is on interpretable (meaningful) and applicable (useful) models. In business problems, very often, several of the questions from the list above must be answered, requiring several different analysis approaches. Therefore we can characterize data mining as a repetitive process of intelligent data analysis.

Neuro-fuzzy methods as they are described in this thesis can be applied in data mining mainly in the areas of classification and prediction. If the underlying relationships are simple enough, neuro-fuzzy methods can also be useful for concept description and in rare cases also for a simple form of dependency analysis, where influential variables must be selected from several independent variables. However, usually in both areas, more complex relationships must be described and graphical models like Bayesian networks [Pearl, 1988] or possibilistic networks [Dubois and Prade, 1988, Gebhardt and Kruse, 1994a, Gebhardt and Kruse, 1994b] are much more useful.

Chapter 3

Neuro-Fuzzy Systems

In this chapter we discuss the notion of a *neuro-fuzzy system* and provide a definition that will be used throughout the thesis. A neuro-fuzzy system is essentially a fuzzy system that uses learning techniques known from artifical neural networks to modifiy its own parameters. Therefore we present some foundations of fuzzy systems (Section 3.1) and neural networks (Section 3.2) before we discuss some fundamental issues in neuro-fuzzy systems in Section 3.3. Section 3.4 discusses the interpretability in fuzzy systems, which are a key factor in the application of neuro-fuzzy methods in data analysis.

3.1 Fuzzy Systems

In this section some foundational issues of fuzzy systems are discussed. For a complete introduction refer, for example, to [Kruse et al., 1994a, Zimmermann, 1996].

A fuzzy set [Zadeh, 1965] is usually identified by its characteristic funtion or membership funtion.

Definition 3.1 *A fuzzy set μ of X is a mapping from the set X to the unit interval*

$$\mu : X \to [0, 1].$$

$\mu(x)$ is called a degree of membership. $\mathcal{F}(X)$ denotes the set of all fuzzy sets of X. We also use the following terms to describe a fuzzy set:

- $[\mu]_\alpha = \{x | x \in X \wedge \mu(x) \geq \alpha\}$ *is called the α-cut of μ,*
- $\text{support}(\mu) = [\mu]_0$ *is called the support of μ,*
- $\text{core}(\mu) = [\mu]_1$ *is called the core of μ.*

Fuzzy sets can be conveniently used to represent linguistic expressions like *approximately zero, small, large*, etc. Fuzzy sets are therefore usually labelled. Most applications that are based on fuzzy sets use simple parameterized membership functions like triangular, trapezoidal or bell-shaped functions. Usually for each considered variable a set of fuzzy sets is specified. We use the term *fuzzy partition* to denote such a set of fuzzy sets. Fuzzy partitions are often specified such that they cover the whole domain of the corresponding variable and that for each value of the domain the degrees of membership to fuzzy sets add up to 1. A typical fuzzy partition of triangular and trapezoidal fuzzy sets is shown in Figure 3.1.

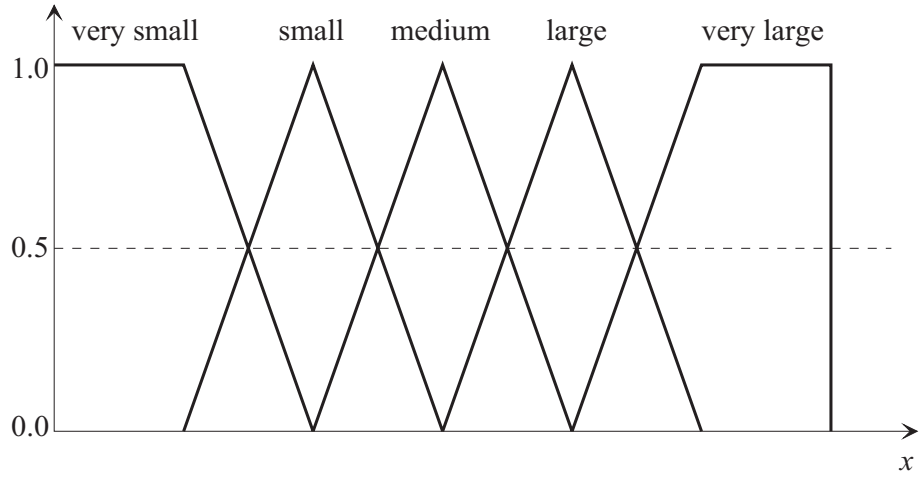


Figure 3.1: A typical fuzzy partition

If fuzzy sets are used to describe relations between variables we obtain linguistic rules or fuzzy rules. A system of several fuzzy rules is called a *fuzzy system*. A fuzzy rule has the form

$$R_r\text{: if } x_1 \text{ is } \mu_r^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \mu_r^{(n)} \text{ then } y \text{ is } \nu_r.$$

Usually the fuzzy sets are replaced by their labels, which results in more readable rules like, for instance,

$$R_r\text{: if } x_1 \text{ is } \textit{small} \text{ and } \ldots \text{ and } x_n \text{ is } \textit{large} \text{ then } y \text{ is } \textit{approximately zero}.$$

A fuzzy rule can also have more than one variable in its consequent.

For the fuzzy systems considered in this thesis, a fuzzy rule must not be interpreted in the sense of an implication, but as a part of the definition of a function known only at some points. The antecedent describes a vague environment and the consequent provides a vague description of the value that is assumed by the output variable y, if the input vector $(x_1, \ldots, x_n)$ lies within the vague environment described by the antecedent.

A fuzzy system uses a set of such fuzzy rules and provides a computational scheme describing how the rules must be evaluated and combined to compute a crisp output value (vector) for any crisp input vector. One can therefore think of a fuzzy system simply as a parameterized function that maps real vectors to real vectors.

Definition 3.2 *A fuzzy system $F_{\mathcal{R}}$ is a mapping*

$$F_{\mathcal{R}} : X \to Y,$$

where $X = X_1 \times \ldots \times X_n \subseteq \mathbb{R}^n$ is called a domain or input space, $Y = Y_1 \times \ldots \times Y_m \subseteq \mathbb{R}^m$ is called a co-domain or output space and $\mathbf{x} = (x_1, \ldots, x_n) \in X$ and $\mathbf{y} = (y_1, \ldots, y_m) \in Y$ denote an input vector and an output vector, respectively. $\mathcal{R}$ is a fuzzy rule base that determines the structure of the fuzzy system:

$$\mathcal{R} = \{R_1, \ldots, R_r\}.$$

Each rule $R_k \in \mathcal{R}$ is a tuple of fuzzy sets

$$R_k = (\mu_k^{(1)}, \ldots, \mu_k^{(n)}, \nu_k^{(1)}, \ldots, \nu_k^{(m)}),$$

where $\mu_k^{(i)}$ is a fuzzy set over the domain of input variable x_i and $\nu_k^{(j)}$ is a fuzzy set over the domain of output variable y_j. We define

$$F_{\mathcal{R}}(\mathbf{x}) = \mathbf{y} = (y_1, \ldots, y_m),$$

where

$$y_j = \text{defuzz}\left(\bigperp_{R_k \in \mathcal{R}} \left\{ \hat{\nu}_k^{(j)} \right\} \right), \quad \text{with}$$

$$\hat{\nu}_k^{(j)} : Y_j \to [0,1],\ y_j \mapsto \top_2 \left\{ \tau_k,\ \nu_k^{(j)} \right\}, \quad \text{with}$$

$$\tau_k = \top_1 \left\{ \mu_k^{(1)}(x_1), \ldots, \mu_k^{(n)}(x_n) \right\}$$

where $\top_1$ and $\top_2$ are t-norms, $\perp$ is a t-conorm, τ_k is the degree fulfilment of fuzzy rule R_k and defuzz *is a so-called defuzzification method*

$$\text{defuzz} : \mathcal{F} \to \mathbb{R}$$

that is used to convert an output fuzzy set $\hat{\nu}_k^{(j)}$ into a crisp output value.

The t-norm $\top_1$ is used to implement the and-operation that conjunctively combines the degrees of membership of the input values to the degree of fulfilment τ_k. For this t-norm, usually the min-operation is used. The t-norm $\top_2$ implements the inference-operation to compute the conclusion of the rule. The two most often used variants of fuzzy systems are the *Mamdani-type* fuzzy system [Mamdani and Assilian, 1975] and the *Sugeno-type* fuzzy system [Sugeno, 1985, Takagi and Sugeno, 1985].

Definition 3.3 *A Mamdani-type fuzzy system $MF_{\mathcal{R}}$ is a fuzzy system with*

(i) $\top_1\{a,b\} = \min\{a,b\}$

(ii) $\top_2\{a,b\} = \min\{a,b\}$

(iii) $\perp\{a,b\} = \max\{a,b\}$

A suitable defuzzification method in Mamdani-type fuzzy systems is, for example, the center-of-gravity method or the mean-of-maximum method [Kruse et al., 1994a]. The evaluation procedure of a Mamdani-type fuzzy system is also known as *max-min-inference*. Sometimes the product is used for the t-norm $\top_1$. In this case the evaluation procedure is known as *max-dot-inference*.

Definition 3.4 *A Sugeno-type fuzzy system $SF_{\mathcal{R}}$ is a fuzzy system that uses special kinds of fuzzy rules. Each rule R_k of the rule base $\mathcal{R}$ is a tuple*

$$R_k = (\mu_k^{(1)}, \ldots, \mu_k^{(n)}, f_k^{(1)}, \ldots, f_k^{(m)}),$$

where $\mu_k^{(i)}$ is a fuzzy set over the domain of input variable x_i and $f_k^{(j)} : X \to Y_j$ is a function over the input variables to determine the value of output variable y_j. With $\mathbf{x} \in X$, $SF_{\mathcal{R}}(\mathbf{x})$ is given as follows

$$SF_{\mathcal{R}}(\mathbf{x}) = \mathbf{y} = (y_1, \ldots, y_m),$$

with

$$y_j = \frac{\sum\limits_{r \in \mathcal{R}} \prod\limits_{i=1}^{n} \mu_r^{(i)}(x_i) \;\cdot\; f_r^{(j)}(\mathbf{x})}{\sum\limits_{r \in \mathcal{R}} \prod\limits_{i=1}^{n} \mu_r^{(i)}(x_i)}.$$

The functions $f_r^{(j)}$ used in the consequents of the rules of a Sugeno-type fuzzy system are local models. $f_r^{(j)}(\mathbf{x})$ defines the contribution of r to the overall output value of $SF_{\mathcal{R}}(\mathbf{x})$ if $\mathbf{x}$ is from the vague environment described by the antecedent of r. Usually linear local models are used:

$$f_r^{(j)}(\mathbf{x}) = \left(a_0^{(j,r)} + \sum_{i=1}^{n} a_i^{(j,r)} x_i \right).$$

Fuzzy systems as they are introduced by the previous three definitions are linguistic representations of piecewise defined functions. The evaluation of the rule base provides an interpolation strategy in a vague environment. The inputs and outputs are crisp values. Only the internal computation is fuzzy.

Fuzzy systems can also be used for classification problems which can be interpreted as a special case of function approximation. In a crisp classification problem an input vector (pattern) must be assigned to one of several classes. A class is a subset of the pattern space. A fuzzy classification problem accepts that a pattern is assigned to several classes with different degrees of membership. In this case a class is a fuzzy set of the pattern space. A classification problem can be easily be transformed into a function approximation problem by specifiying a set $\tilde{\mathcal{L}}$ with patterns $(\mathbf{x}, \mathbf{c})$, where $\mathbf{c} \in [0,1]^m$ and c_i denotes the degree of membership of $\mathbf{x}$ in class C_i. If a crisp classification problem is represented this way, then in each $\mathbf{c}$ there is exactly one component set to 1 and all other components are set to 0. A fuzzy system used for classification is called a fuzzy classifier.

Definition 3.5 *A fuzzy classifier is a fuzzy system*

$$F_{\mathcal{R}} : X \to Y,$$

with $Y = [0,1]^m$. *Its rule base* $\mathcal{R}$ *consists of special kinds of fuzzy rules of the form*

$$R_k = (\mu_k^{(1)}, \ldots, \mu_k^{(n)}, c_{j_k}),$$

where $c_{j_k} \in C = \{c_1, \ldots, c_m\}$ *is a class label. We define*

$$F_{\mathcal{R}}(\mathbf{x}) = \mathbf{y} = (y_1, \ldots, y_m),$$

with

$$y_j = \mathop{\perp}_{\substack{R_k \in \mathcal{R} \\ \mathrm{con}(R_k) = c_j}} \{\tau_k\}$$

where $\perp$ *is a t-conorm and* $\mathrm{con}(R_k)$ *is the consequent of rule* R_k.

The output of a fuzzy classifier is a vector whose components denote the degree of membership of a processed pattern to the available classes. In many applications a pattern must be assigned to a single class only. In this case the output vector of a fuzzy classifier must be interpreted (or defuzzified). Usually a "winner takes all" interpretation is used, i.e. the class with the largest degree of membership is selected.

We interpret fuzzy systems as convenient models to linguistically represent (non-linear) mappings [Zadeh, 1996]. The designer of a fuzzy system specifies characterisitc points of an assumed underlying function. This function is unknown except for those characteristic points. The fuzzy sets that are used to linguistically describe those points express the degree of indistinguishability of points that are close to each other. Fuzzy systems can be interpreted on the basis of equality relations [Kruse et al., 1994a, Klawonn et al., 1995a]. This interpretation also provides a formal justification of Mamdani's approach.

The advantages of applying a fuzzy system are the simplicity and the linguistic interpretation of the approach. This allows for the inexpensive and fast development and maintenance of solutions and thus enables us to solve problems in application areas where rigorous formal analysis would be too expensive and time-consuming.

Fuzzy systems also have information compression capabilities. The fuzzy partitions of the variables of a considered problem specify a certain granularity under which data is processed [Zadeh, 1994a, Zadeh, 1994b]. The user can decide which areas of the domain are less important for solving the problem under consideration and can be treated with a coarse granularity. Important areas are processed with finer granularity. The local approach of fuzzy systems also allows us to ignore areas of the domain completely where no data is located that is relevant to the problem.

The design process of a fuzzy system is based on knowledge acquisition. It requires an expert to specify fuzzy sets to partition all variables and a sufficient number of fuzzy rules to describe the input/output relation of the problem of interest. Fuzzy sets can be created by providing characteristic values for all variables and equality relations that determine the degree of indistinguishability. The indistinguishability is, on the one hand, required due to insufficient precision of measured input values and it is, on the other hand, desired by the expert because there is no need to distinguish values with small differences. The specified equality relations induce fuzzy sets (fuzzy singletons) at the provided characteristic values and each tupel of characteristic values constitutes a rule. For a complete description on how to design fuzzy systems on the basis of equality relations see [Kruse et al., 1994a].

A fuzzy system that is constructed by knowledge acquisition alone, will usually not perform as required when it is applied. A manual *tuning process* must usually be appended to the design stage. This is because the expert can be wrong about the location of the specified characteristic points, the number of rules, or the degree of indistinguishability in certain areas of the data space. The tuning process results in modifying the membership functions and/or the rule base of the fuzzy system.

This tuning process can be very time-consuming and error-prone. It is therefore useful to support the design process of fuzzy systems by automatic learning approaches that can make use of available data samples. One possible way to estimate the parameters of a fuzzy system based on training data is to use learning methods that are derived from artifical neural networks. Before we consider how to apply such learning techniques we provide some fundamental notations on neural networks

3.2 Neural Networks

An (artificial) neural network implements a mapping from real vectors to real vectors [Haykin, 1994, Rojas, 1996, Zell, 1994, Zurada, 1992]. For this purpose they use a network of simple interconnected processing units (formal neurons) that

obtain weighted inputs and compute a single output value via a so-called activation function. The connection weights are estimated by learning algorihtms, for example by backpropagation [Werbos, 1974, Rumelhart et al., 1986a, Rumelhart et al., 1986b] or its variants like quickpropagation or resilient propagation [Zell, 1994].

Learning algorithms of neural networks use a *learning problem* described by a set of training data and iteratively update the parameters of a network such that some error measure is decreased or some performance measure is increased. The training data can consist of input and output data (supervised learning), of input data and success or failure signals (reinforcement learning), or of input data alone (unsupervised learning). If the training data also contains output data, we speak of a *fixed learning* problem, otherwise we use the term *free learning problem*.

Definition 3.6 *A fixed learning problem is given by a set*

$$\tilde{\mathcal{L}} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_s, \mathbf{y}_s)\},$$

with an input pattern (vector) $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,n}) \in X = X_i \times \ldots \times X_n$ *and a target output pattern (vector)* $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,m}) \in Y = Y_1 \times \ldots \times Y_m$ *A pair* $(\mathbf{x}, \mathbf{y})$ *is called training pattern. The set* X *is called domain or input set and the set* Y *is called co-domain or output set.* $X \times Y$ *is called universe of discourse. A free learning problem consists only of input patterns and is given by a set*

$$\mathcal{L} = \{\mathbf{x}_1, \ldots, \mathbf{x}_s\}.$$

Remark 3.7 The nature of the input set and the output set of a learning problem depends on the kind of model that is created by learning. For the rest of this section we assume that for neural networks $X \subseteq \mathbb{R}^n$ and $Y \subseteq \mathbb{R}^m$ holds.

A neural network can be generally seen as a formal structure that can be described by a set and a few mappings. The following definition provides a generic model that covers most neural network architectures [Nauck et al., 1997].

Definition 3.8 *A neural network is a tuple* $(U, W, A, O, \mathrm{NET}, \mathrm{ex})$*, where*

(i) U *is a finite set of processing units (neurons),*

(ii) W*, the network structure, is a mapping from the Cartesian product* $U \times U$ *to* $\mathbb{R}$*,*

(iii) A *is a mapping that assigns an activation function* $A_u : \mathbb{R}^3 \to \mathbb{R}$ *to each* $u \in U$*,*

(iv) O *is a mapping that assigns an output function* $O_u : \mathbb{R} \to \mathbb{R}$ *to each* $u \in U$*,*

(v) NET *is a mapping that assigns a network input function* $\mathrm{NET}_u : (\mathbb{R} \times \mathbb{R})^U \to \mathbb{R}$ *to each* $u \in U$*, and*

(vi) ex *is an external input function* $\mathrm{ex} : U \to \mathbb{R}$*, that assigns to each* $u \in U$ *an external input in the form of a real value* $\mathrm{ex}_u = \mathrm{ex}(u) \in \mathbb{R}$*.*

To approximate a given function a neural network must have a suitable network structure and the units must use suitable activation and output functions. The network structure is often given in the form of a matrix (connection matrix, weight matrix) where each entry specfies the connection strength (weight or synaptic weight) between two units. If for any two units u, v we have $W(u, v) = 0$, we say that those units are not connected. By setting certain weights to zero neural networks can assume arbitrary structures. Usually layered structures are used, where there are one input and one output layer and several hidden (inner) layers. Connections within layers or between non-consecutive layers are not commonly used. Such networks are also called *feed-forward networks* or *perceptrons*. In the case of layered network architectures the network structure is usually given by several connection matrices, one for each two interconnected layers.

Based on this generic model we can describe the two most common neural network architectures used in applications – multilayer perceptrons (MLP) and radial basis function networks (RBFN).

Definition 3.9 *A multilayer perceptron is a neural network* $MLP = (U, W, A, O, \mathrm{NET}, \mathrm{ex})$ *that has the following properties:*

(i) $U = U_1 \cup \ldots \cup U_n$ *is a set of processing units (neurons) where* $n \geq 3$ *is assumed. Furthermore,* $U_i \neq \emptyset$ *for all* $i \in \{1, \ldots, n\}$ *and* $U_i \cap U_j = \emptyset$ *for* $i \neq j$*.* U_1 *is called the input layer and* U_n *the output layer. The* U_i *with* $1 < i < n$ *are called hidden layers.*

(ii) The network structure is given by the function $W : U \times U \to \mathbb{R}$*. There exist only connections between consecutive layers.*
Thus, $(u \in U_i \ \wedge \ W(u, v) \neq 0) \implies v \in U_{i+1}$ *for all* $i \in \{1, \ldots, n-1\}$*.*

(iii) A assigns an activation function $A_u : \mathbb{R} \to \mathbb{R}$ *to each unit* $u \in U$ *to calculate the activation* a_u *with*

$$a_u = A_u(\mathrm{ex}(u)) = \mathrm{ex}(u)$$

for all $u \in U_1$*, and*

$$a_u = A_u(\mathrm{net}_u) = f(\mathrm{net}_u)$$

for all $u \in U_i$ $(i \in \{2, \ldots, n\})$*, where all units use the same non-linear function* $f : \mathbb{R} \to [0, 1]$*.*

(iv) O *assigns an output function* $O_u : \mathbb{R} \to \mathbb{R}$ *to each unit* $u \in U$ *to calculate the output* o_u*, with* $o_u = O_u(a_u) = a_u$ for all $u \in U$.

(v) NET *assigns a network input (propagation) function* $\mathrm{NET}_v : (\mathbb{R} \times \mathbb{R})^{U_{(i-1)}} \to \mathbb{R}$ *to each unit* $v \in U_i$ $(2 \leq i \leq n)$ *to compute the network input* net_v*, with*

$$\mathrm{net}_v = \sum_{u \in U_{i-1}} W(u, v) \cdot o_u + \theta_v.$$

$\theta_v \in \mathbb{R}$ *is the bias of unit* v.

(vi) $\mathrm{ex} : U_1 \to \mathbb{R}$ *assigns an external input* $\mathrm{ex}_u = \mathrm{ex}(u)$ *to each input unit* $u \in U_1$.

The hidden units must use a non-linear activation function f. Multilayer systems that consist only of linear units can always be represented by a network without hidden layers. The output vector of a multilayer neural network is obtained by several consecutive vector–matrix multiplications, each followed by an application of the activation function. If the hidden units used a linear activation function, all connection matrices could be multiplied into a single matrix, thus obtaining a single-layer network.

The activation of the neurons is usually from the set $[0, 1]$ or $[-1, 1]$. However, any other real interval is also possible. Neurons with activations from, for example, $\{0, 1\}$ can be used for the model as well. If we restrict ourselves to a multilayer perceptron with such binary units the function f can be chosen as a linear threshold function as with the simple perceptron. In general, however, real-valued units as described in the definition are used and, therefore, a continuous activation function is necessary. The learning algorithm for multilayer perceptrons requires a differentiable activation function.

The following sigmoid (S-shaped) functions are most frequently used as activation functions (with $\alpha, \beta > 0$):

- $f(x) = \dfrac{\alpha}{1 + \mathrm{e}^{-\beta x}}$ (logistic function, asymptotically approaches 0 and α)
- $f(x) = \alpha \cdot \tanh(\beta x) = \alpha \cdot \dfrac{\mathrm{e}^{\beta x} - \mathrm{e}^{-\beta x}}{\mathrm{e}^{\beta x} + \mathrm{e}^{-\beta x}}$ (asymptotically approaches $-\alpha$ and α),
- $f(x) = \frac{\alpha}{\pi}(\frac{\pi}{2} + \arctan(\beta x))$ (asymptotically approaches 0 and α).

In all cases the steepness is determined by the parameter β.

It was shown that an MLP is a universal function approximator [Funahashi, 1989, Hecht-Nielsen, 1989, Hecht-Nielsen, 1990, Hornik et al., 1989, Hornik et al., 1990, White, 1990], i.e. an MLP with a potentially infinite number

of hidden units that are organized within one hidden layer can approximate any bounded continous function to any given degree of accuracy. In designing an MLP for a given problem a suitable number of hidden units must be specified. If this number is too small the MLP might not be able to solve the problem. If the number of hidden units is too large, the MLP does not generalize well and tends to "memorize" the training patterns instead. This means, the performance of the MLP on the training set is very good, whereas the performance on previously unseen data is poor.

Each hidden unit of an MLP divides its input space by a hyperplane. The output that is generated for an input vector depends on its relative position (and distance) to all hyperplanes created by the network. This kind of procedure corresponds to a *global view* – all points on the same side of a hyperplane belong to the same class with regard to this special hyperplane.

An RBFN treats the input patterns locally. Each hidden unit uses a radial basis function to compute its activation, and thus encompasses a subset of the input space. An RBFN is based on the interpolation or approximation of a funciton by the superposition of radial basis functions [Poggio and Girosi, 1989].

To interpolate an arbitrary continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ samples $\mathbf{x}_i \in \mathbb{R}^n$ are needed such that $f(\mathbf{x}_i) = d_i$ is known. If s samples are given, an interpolation with s basis functions is done as follows:

$$f(\mathbf{x}) = \sum_{i\,=\,1}^{s} w_i \cdot h_{\mathbf{x}_i}(\mathbf{x}),$$

with s arbitrary basis functions $h_{\mathbf{x}_i} : \mathbb{R}^n \rightarrow \mathbb{R},\ i = 1, \ldots, s$. The centers of the basis functions are the samples $\mathbf{x}_i$. Often a radial-symmetric function like

$$h_{\mathbf{x}_i}(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}||\mathbf{x} - \mathbf{x}_i||^2\right)$$

is used, where $||\cdot||$ is an arbitrary vector norm, e.g. the Euclidean norm

$$||\mathbf{x}|| = \sqrt{\sum_{j=1}^{n} x_j^2},\ \text{with } \mathbf{x} = (x_1, \ldots, x_n).$$

The solution of an interpolation problem like this is to specify a function f that assumes the given values d_i at the s samples $\mathbf{x}_i$. For this s basis functions are chosen, whose centers are given by the samples. The still undetermined factors w_i are found by solving a system of linear equations:

$$\mathbf{H} \cdot \mathbf{w} = \mathbf{d} \Longleftrightarrow \begin{bmatrix} h_{11} & \ldots & h_{1s} \\ \vdots & \vdots & \vdots \\ h_{s1} & \ldots & h_{ss} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ \vdots \\ w_s \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_s \end{bmatrix},$$

with $h_{ij} = h_{\mathbf{x}_i}(\mathbf{x}_j)$. If all s supporting points are different from each other, the matrix $\mathbf{H}$ is positive definite, and the system of equations has the unique solution $\mathbf{w} = \mathbf{H}^{-1} \cdot \mathbf{d}$.

If not all samples are used as centers of basis functions but only a subset, we obtain a *function approximation problem*, where we do not require that the function f should assume the given values d_i at the samples $\mathbf{x}_i$. In this case, the matrix $\mathbf{H}$ is no longer quadratic and thus the resulting system of equations is over-determined. In this case the factors w_i are determined by an approximate solution with the help of the pseudo-inverse $\mathbf{H}^+$ of $\mathbf{H}$:

$$\mathbf{w} = \mathbf{H}^+ \cdot \mathbf{d} \;=\; (\mathbf{H}^T \cdot \mathbf{H})^{-1} \cdot \mathbf{H}^T \cdot \mathbf{d}.$$

An RBFN encodes such an intperpolation/approximation strategy in a three-layer feed-forward neural network architecture.

Definition 3.10 *A radial basis function network (RBFN) is a neural network* $(U, W, A, O, \mathrm{NET}, \mathrm{ex})$ *with the network topology of an MLP and the following specifications:*

(i) $U = U_I \cup U_H \cup U_O$, *with input layer* $U_I = \{u_1, \ldots, u_n\}$, *hidden layer* $U_H = \{v_1, \ldots, v_r\}$ *and output layer* $U_O = \{z_1, \ldots, z_m\}$. *The number* r *of hidden units is not larger than the number* s *of training patterns of a fixed learning problem* $\tilde{\mathcal{L}}$ *which is given for the RBFN.*

(ii) The weights $W(u, v)$, *with* $u \in U_I$ *and* $v \in U_H$, *are given by* $r \leq s$ *arbitrary but different input patterns of the learning problem* $\tilde{\mathcal{L}}$: $W(u_k, v_i) = x_{j_i,k}$, *with* $k \in \{1, \ldots, n\}$, $i \in \{1, \ldots, r\}$, $j_i \in \{1, \ldots, s\}$.

(iii) A assigns an activation function to each unit $u \in U$. *For all input units* $u \in U_I$ *the activation function* A_u *is used to calculate the activation* a_u:

$$a_u = A_u(\mathrm{ex}_u) = \mathrm{ex}_u.$$

For all hidden units $v \in U_H$ *the activation function* A_v *to calculate the activation* a_v *is*

$$a_v = A_v(\mathrm{net}_v) = h(\mathrm{net}_v),$$

where h *is a radial basis function in* $\mathbb{R}^1$ *with* 0 *as its center.*
For all output units $w \in U_O$ *a linear activation function* A_w *is employed to determine the activation*

$$a_w = A_w(\mathrm{net}_w) = \mathrm{net}_w.$$

(iv) O *assigns an output function* O_u *to each unit* $u \in U$ *to calculate the output* o_u*, with* $o_u = O_u(a_u) = a_u$*.*

(v) NET *assigns to each unit* $v \in U_H$ *and each unit* $w \in U_O$ *a network input function (propagation function)* NET_v *and* NET_w*, respectively, to calculate the network input* net_v *and* net_w*. For hidden units* $v \in U_H$ *the network input is*

$$\mathrm{net}_v = ||\mathbf{o} - \mathbf{w}_v||$$

where $\mathbf{o} = (o_{u_1}, \ldots, o_{u_n})$ *is the output vector of the input layer and* $\mathbf{w}_v = (W(u_1, v), \ldots, W(u_n, v))$ *is the weight vector of the hidden unit* v*. For all output units* $w \in U_O$ *the network input is*

$$\mathrm{net}_w = \sum_{v \in U_H} o_v \cdot W(v, w) + \theta_w,$$

where $\theta_w \in \mathbb{R}$ *is the bias of unit* w*.*

(vi) $\mathrm{ex} : U_I \to [0, 1]$ *assigns to each input unit* $u \in U_I$ *its external input* $\mathrm{ex}_u = \mathrm{ex}(u)$*.*

Remark 3.11 To obtain the structure of a neural network that complies with Definition 3.8, Definition 3.10 splits up the multidimensional radial basis functions into the computation of a vector norm and the computation of the distance from the center point. The vector norm is calculated by the propagation function, and the distance is determined by the activation function of a hidden neuron. The center of the multidimensional RBF is encoded as the weight vector of a hidden unit.

However, from a more pragmatic point of view we can say that each hidden unit represents a multidimensional RBF, without always explicitly considering that it is actually represented in a distributed way.

If $r = s$ holds for an RBFN, we speak of a *simple RBFN*. A simple RBFN creates for each sample (training pattern) a hidden neuron (a basis function). In this case, a system of linear equations can be solved for each output neuron and the solutions determine the weights to the output layer. If with s training patterns s hidden units are used, the result is an interpolation function that matches the given support points. Obviously, this procedure is not appropriate for larger learning problems, since the solution of the system of equations requires, for s training patterns, the inversion of an $s \times s$ matrix. This incurs polynomial computation costs in the order of $O(s^3)$.

Besides this, training data is usually noisy. Therefore one is less interested in an interpolation function but more interested in a sufficient generalization of the training data and a suitable approximation function. For this reason, an RBFN usually uses $r \ll s$ hidden neurons and instead of interpolating the desired function it is

restricted to approximating it. In this case the weight vector can be determined by an approximate solution with the help of the pseudo-inverse.

For an RBFN it is necessary to select the centers of the basis functions. Definition 3.10 demands that r arbitrary but different training patterns are chosen. This is similar to the design problem of an MLP, where a suitable number of hidden units must be determined. In the case of an RBFN not only the specification of r but also the initialisation of the centers is difficult. A learning algorithm can update the centers and weights of an RBFN during a training stage. However, unsuitable centers can delay or even prevent the training success. Therefore the centers are often selected by applying a cluster analysis to the training data first. The cluster centers are used as centers for the radial basis functions.

For both MLP and RBFN, backpropagation is the most important learning method to determine the parameters of a neural network. In an MLP the weights are updated and in an RBFN the centers of the radial basis functions and the weights to the output units are trained. We interpret backpropagation as an "idea of a learning algorithm" and not as a specific implementation. Backpropagation is not only restricted to neural networks but can also be applied to any parameterized input/ouput system that computes outputs in a distributed way.

Backpropagation is a method to construct a learning algorithm to iteratively update the parameters of an input/output system. An input/output system computes an output and provides it at its output interface. The computation takes place by processing some input provided at the input interface of the system. Backpropagation (BP) contains the following steps:

(i) Determine the current output by propagating the current input (forward) through the system from the input interface to the output interface (forward path).

(ii) Determine an error or performance signal given the current output of a system.

(iii) Propagate the error/performance signal backwards through the system by distributing it via every path from the output interface to the input interface (backward path).

(iv) For each parameter of the system that is reached during the backpropation of the error/performance signal, locally compute

  (a) a modification of this parameter and

  (b) a modified error/performance signal that is backpropagated further.

(v) Repeat steps (i)–(iv) until the error/performance signal assumes a suitable value.

These steps can be summarized as follows: *BP is a method to iteratively and locally compute modifications for the parameters of a system based on an error signal depending on the current output of the system.*

The main feature of the idea of BP is that an error signal is distributed backwards through the system architecture using the same information pathways that are used to compute the output, only in the opposite direction. It is also important that the modifications of the parameters are computed locally. This means that updating one parameter does not depend on updating another paramter and in general allows us to implement BP in a parallel fashion. Of course, implementational constraints and the architecture of the specific system trained by BP will usually result in some sequence in which parameters are updated. It is neither prescribed how to compute the error or performance signal and its modification during backpropagation nor how to determine the modifications of the parameters.

For neural networks like MLP and RBFN BP is usally implemented by a simple gradient descent procedure or some variation of it. The error signal on the output side is given by the difference $(t_u^{(p)} - o_u^{(p)})$, where $t_u^{(p)}$ is a target value for output unit u given by the pth training pattern and $o_u^{(p)}$ is the actual output value of output unit u caused by the progatation of the pth input pattern. The error of the whole network over all training patterns is given by the sum of squared errors (SSE)

$$SSE = \frac{1}{2} \sum_{p \in \tilde{\mathcal{L}}} E^{(p)} = \frac{1}{2} \sum_{p \in \tilde{\mathcal{L}}} \sum_{u \in U_O} (t_u^{(p)} - o_u^{(p)})^2. \tag{3.1}$$

For an MLP the weight updates $\Delta_p W(u,v)$ are computed according to

$$\Delta_p W(u,v) = \eta \frac{\partial E^{(p)}}{\partial W(u,v)},$$

where $\eta \in \mathbb{R}^+$ is called step size or learning rate. In the literature the term "backpropagation" is usually identified with the implemention of BP in the form of a gradient descent procedure. To distinguish our notion of backpropagation from its actual implementation we refer to the implementation given in the following definition as *plain backpropagation* (plain BP).

For an MLP with $n \geq 2$ layers $U = U_1 \cup \ldots \cup U_n$ plain BP is implemented as follows. After the pth input pattern was propagated we compute

$$\Delta_p W(u,v) \;=\; \eta\, \delta_v^{(p)}\, a_u^{(p)}$$

with $u \in U_{i-1},\ v \in U_i,\ 2 \leq i \leq n,\ \eta > 0$ and

$$\delta_v^{(p)} \;=\; \begin{cases} f'(\mathrm{net}_v^{(p)})\,(t_v^{(p)} - a_v^{(p)}) & \text{if } v \in U_n \\ f'(\mathrm{net}_v^{(p)}) \sum\limits_{\tilde{v} \in U_{j+1}} \delta_{\tilde{v}}^{(p)}\, W(v,\tilde{v}) & \text{if } v \in U_j, 2 \leq j \leq n-1, \end{cases}$$

where f is a sigmoid activation function, $a_u^{(p)}$ is the activation of unit u after propagation of the pth input pattern and $t_v^{(p)}$ is the target output (activation) of output unit $v \in U_n$ given by the pth target output pattern.

Instead of plain BP often *backpropagation with momentum* (BP with momentum) is applied, where the modificaton of a weight is computed by

$$\Delta_p W(u,v) = \eta\, \delta_v^{(p)}\, a_u^{(p)} \;+\; \beta\, \Delta_{\text{last}} W(u,v)$$

where $\Delta_{\text{last}} W(u,v)$ is the most recent modification of $W(u,v)$ and $\beta \in \mathbb{R}^+$ is called momentum.

Plain BP is also known as *generalized delta rule* and BP with momentum is also known as *conjugate gradient descent*. For an implementation of BP for RBFN see, for example, [Haykin, 1994, Nauck et al., 1997].

Plain BP and BP with momentum are usually slow learning algorithms and prone to get stuck in local minima or to oscillate. Implementations like quickpropagation, which assume a quadratic shape of the error surface and use second order derivatives or implementations which use individual adaptive learning rates for each weight like resilient propagation are usually much faster [Zell, 1994]. For the rest of the thesis the exact implementations of BP in neural network models is not important, as neuro-fuzzy methods discussed in the following need special learning algorithms that only make use of the general idea of BP, as discussed above. For a discussion of modern approaches in training neural networks see [Neuneier and Zimmermann, 1998].

3.3 Neuro-Fuzzy Systems

The idea of a neuro-fuzzy system is to find the parameters of a fuzzy system by means of learning methods obtained from neural networks. In this chapter the basic properties of neuro-fuzzy systems are discussed. The learning techniques that can be used to create fuzzy systems for data are described in Chapters 4 and 5.

A common way to apply a learning algorithm to a fuzzy system is to represent it in a special neural-network-like architecture. Then a learning algorithm – such as backpropagation – is used to train the system. There are some problems, however. Neural network learning algorithms are usually based on gradient descent methods. They cannot be applied directly to a fuzzy system, because the functions used in the inference process are usually not differentiable. There are two solutions to this problem:

(a) replace the functions used in the fuzzy system (like min and max) by differentiable functions, or

(b) do not use a gradient-based neural learning algorithm but a better-suited procedure.

Modern neuro-fuzzy systems are often represented as multilayer feedforward neural networks. For an overview see, for example, [Buckley and Hayashi, 1994, Buckley and Hayashi, 1995, Jang et al., 1997, Lin and Lee, 1996, Nauck et al., 1997]. Jang's ANFIS model [Jang, 1993], for example, implements a Sugeno-like fuzzy system in a network structure, and applies a mixture of plain backpropagation and least mean squares procedure to train the system. A Sugeno-like fuzzy system uses only differentiable functions, i.e. for ANFIS solution (a) is selected. The GARIC model [Berenji and Khedkar, 1992] also chooses solution (a) by using a special "soft minimum" function which is differentiable. The problem with solution (a) is that the models are sometimes not as easy to interpret as for example Mamdani-type fuzzy systems. Other models like NEFCON [Nauck, 1994b, Nürnberger et al., 1999], NEFCLASS [Nauck and Kruse, 1997b] and NEFPROX [Nauck and Kruse, 1999a] use solution (b) – they are Mamdani-type fuzzy systems and use special learning algorithms.

In addition to multilayer feedforward networks there are also combinations of fuzzy techniques with other neural network architectures, for example self-organizing feature maps [Bezdek et al., 1992, Vuorimaa, 1994], or fuzzy associative memories [Kosko, 1992]. Some approaches refrain from representing a fuzzy system in a network architecture. They just apply a learning procedure to the parameters of the fuzzy system, or explicitly use a neural network to determine them.

There are several different approaches which have much in common, but differ in implementational aspects. To stress the common features of all these approaches, and to give the term *neuro-fuzzy system* a suitable meaning, we only apply it to systems which possess the following properties:

(i) A neuro-fuzzy system is a fuzzy system that is trained by a learning algorithm (usually) derived from neural network theory. The (heuristic) learning procedure operates on local information, and causes only local modifications in the underlying fuzzy system. The learning process is not knowledge-based, but data-driven.

(ii) A neuro-fuzzy system can always (i.e. before, during and after learning) be interpreted as a system of fuzzy rules. It is possible both to create the system out of training data from scratch, and to initialize it from prior knowledge in the form of fuzzy rules.

(iii) The learning procedure of a neuro-fuzzy system takes the semantical properties of the underlying fuzzy system into account. This results in constraints on the possible modifications of the system's parameters.

(iv) A neuro-fuzzy system approximates an n-dimensional (unknown) function that is partially given by the training data. The fuzzy rules encoded within the system represent vague samples, and represent vague prototypes of the training data. A neuro-fuzzy system should not be seen as a kind of (fuzzy) expert system, and it has nothing to do with fuzzy logic in the narrow sense [Kruse et al., 1994a].

(v) A neuro-fuzzy system can be represented by a special three-layer feedforward neural network (see Definition 3.12). This view of a fuzzy system illustrates the data flow within the system and its parallel nature. However, this neural network view is not a prerequisite for applying a learning procedure, it is merely a convenience.

The neuro-fuzzy technique, then, is used to derive a fuzzy system from data, or to enhance it by learning from examples. The exact implementation of the neuro-fuzzy model does not matter. It is possible to use a neural network to learn certain parameters of a fuzzy system, like using a self-organizing feature map to find fuzzy rules [Pedrycz and Card, 1992], or to view a fuzzy system as a special neural network and to apply a learning algorithm directly [Nauck and Kruse, 1996].

A lot of neuro-fuzzy approaches use a neural network-like graph to illustrate the data flow and the computations that are carried out in a fuzzy system. This neural network representation is then used to formalize the application of a learning algorithm. Many neuro-fuzzy approaches use a 5-layer feedforward or *node-oriented* architecture as it is shown in Figure 3.2(a).

We call this representation node-oriented, because in this kind of representation all membership function parameters of the fuzzy system reside inside the nodes of the network and the connections are not needed to carry any parameters. This means, from a neural network point of view, this architecture is not adaptive as long as there are no weights attached to the connections. Such a node-oriented representation of a fuzzy system is therefore often used for defining a neural network-like learning algorithm based on adaptive weights that are attached to some of the connections. This issue is discussed in detail in Section 5.1.

The layers of the network are not fully connected, but the connections are selected such that they represent the rule base of the fuzzy system. The network in Figure 3.2(a) represents a fuzzy system with the following two fuzzy rules:

R_1: if x_1 is *small* and x_2 is *small* then y is *small*
R_2: if x_1 is *large* and x_2 is *large* then y is *large*

The meaning of the layers of the network representation in Figure 3.2(a) is as follows:

- input layer: input variables,

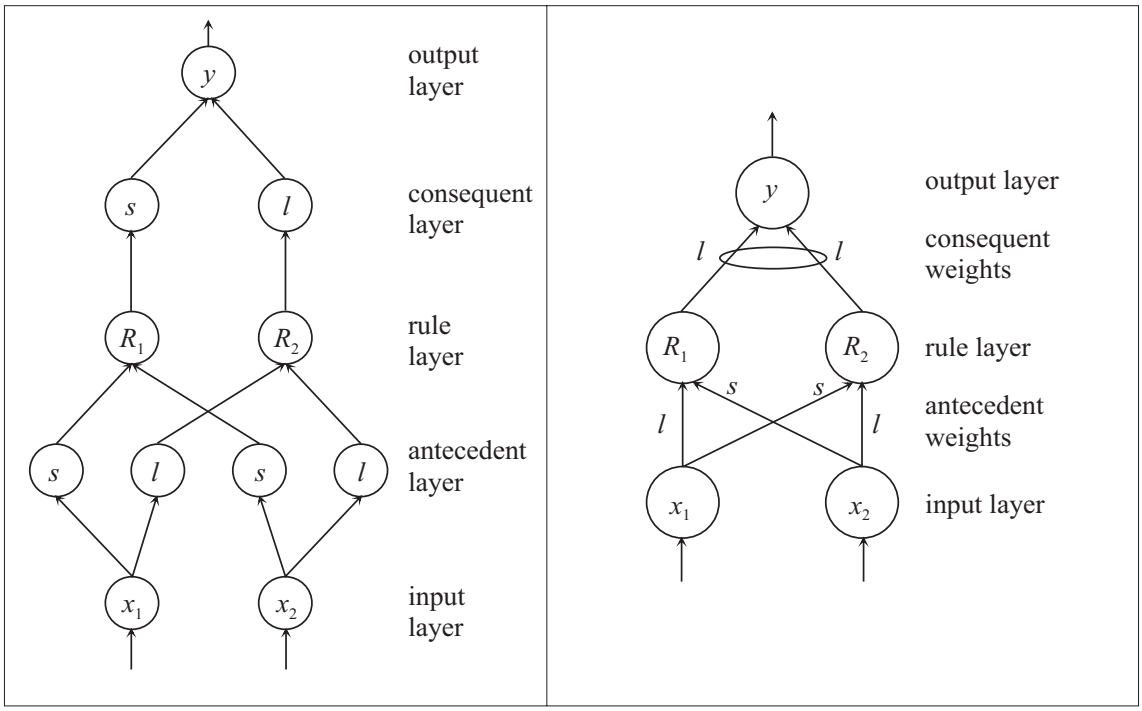


Figure 3.2: Two fuzzy systems represented as a 5-layer feedforward network (a) and as a 3-layer feedforward network with shared weights (b)

- antecedent layer: fuzzy sets used in linguistic terms of antecedents,
- rule layer: fuzzy rules,
- consequent layer: fuzzy sets used in linguistic terms of consequents,
- output layer: ouput variables.

If a learning algorithm is applied to modify parameters of the fuzzy system, this would mean that parameters inside the nodes of the second and fourth layer are modified. By storing the membership functions inside these layers, we ensure that all fuzzy rules that are represented by the nodes of the third layer use the same set of membership functions to represent linguistic terms.

Instead of the 5-layer network representation of a fuzzy system shown in Figure 3.2(a), we prefer a 3-layer or *connection-oriented* representation as it is shown in Figure 3.2(b), where the connections carry fuzzy sets as weights. This representation better corresponds with a neural network view where all parameters that can be changed by a learning algorithm are located at the connections. In order to ensure that each linguistic term is only represented by one fuzzy set, we use *shared weights* (coupled connections).

The fuzzy system in Figure 3.2(b) consists of the two rules

if x_1 is *large* and x_2 is *small* then y is *large*
if x_1 is *small* and x_2 is *large* then y is *large*

In this example, the connections from the rule layer to the output layer share the same (fuzzy) weight – *large* – which is represented as a fuzzy set over the domain of the output variable y. A learning algorithm will recognise that the connections are coupled and make sure that the weight is always identical for both connections.

The meaning of the layers of the network representation in Figure 3.2(b) is equivalent to the layers in Figure 3.2(a) with the same names. In the 3-layer representation the connections also encode parameters of a fuzzy system:

- the weights on the antecedent connections represent fuzzy sets used in linguistic terms of antecedents,
- the weights on the consequent connections represent fuzzy sets used in linguistic terms of consequents.

The networks in Figure 3.2 represent Mamdani-type fuzzy systems. This kind of network representation can also be used to represent a simplified form of Sugeno-type fuzzy systems, where the consequents consist of singletons instead of linear combinations of the input variables. In this case the consequent weights in connection-oriented network representation become singletons. The node-oriented network representation could be used by storing the singletons in the nodes of the consequent layer. However, this is usually not done, but the consequent layer is deleted such that a 4-layer representation is created and the singletons are represented as weights between rule layer and output layer. Such a 4-layer node-oriented representation is used in [Siekmann et al., 1999] to predict the German stock index DAX by implementing it in a neural network development software.

To represent a Sugeno-type fuzzy system that uses linear models in the consequents of its rules, a network representation would require direct connections from the input nodes to the output nodes.

If a fuzzy classifier must be represented in feedforward network, then the network types shown in Figure 3.3 can be used. Because fuzzy classifiers do not use fuzzy sets in the consequents of their rules, the rule nodes can be directly connected via unweighted connections (or via connections with the constant weight 1) to the output nodes which represent class labels. As each rule uses only one class label in its consequent there is exactly one connection protruding from each rule node to the output layer.

The node-oriented network representation in Figure 3.3(a) encodes a fuzzy classifier with the two rules

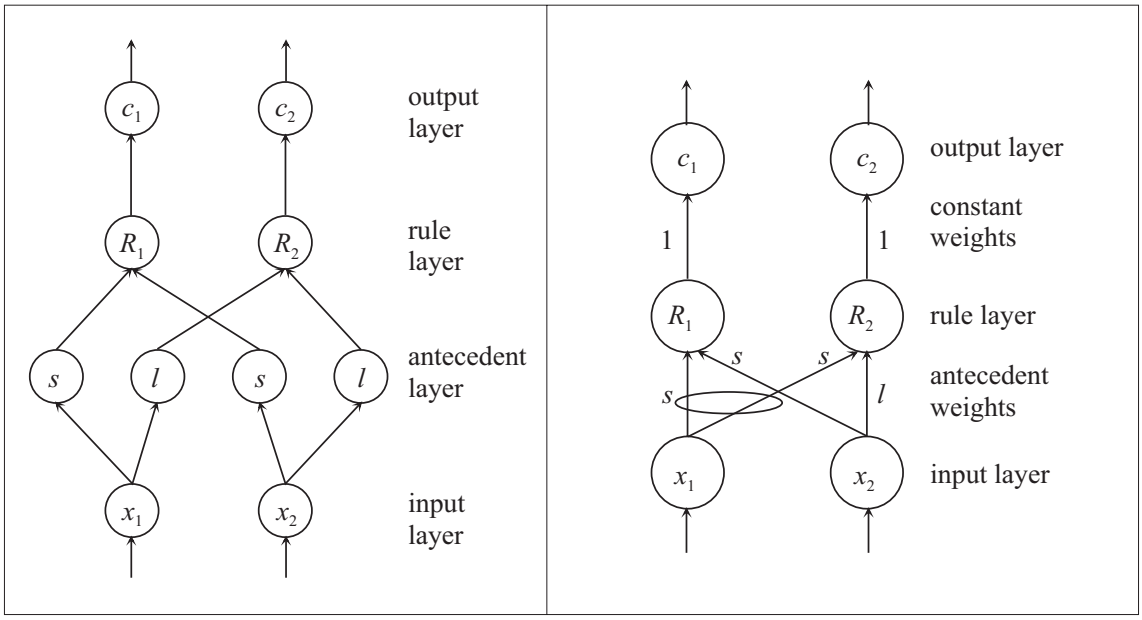


Figure 3.3: Different network representations of two fuzzy classifiers with two rules and two classes
R_1: if x_1 is *small* and x_2 is *small* then class c_1
R_2: if x_1 is *large* and x_2 is *large* then class c_2

and the connection-oriented network representation in Figure 3.3(b) consists of the two rules

R_1: if x_1 is *small* and x_2 is *small* then class c_1
R_2: if x_1 is *small* and x_2 is *large* then class c_2

Note that the connections $x_1 \rightarrow R_1$ and $x_1 \rightarrow R_2$ share the fuzzy set *small* as a common weight.

A definition for a *3-layer fuzzy perceptron* is given below. It provides a way to represent a fuzzy system as a connection-oriented network. The name refers to the structure of the model that is similar to an MLP. The term *fuzzy (multilayer) perceptron* has also been used by other authors for their approaches [Keller and Tahani, 1992, Mitra and Kuncheva, 1995, Pal and Mitra, 1992]. Here this notion is used to describe the topology.

From a neural network point of view a fuzzy perceptron has the architecture of a common multilayer perceptron, but the weights are modeled as fuzzy sets and the activation, output, and propagation functions are changed accordingly, to implement a common fuzzy inference path.

Definition 3.12 *Let $F_{\mathcal{R}}$ be a fuzzy system. A 3–layer fuzzy perceptron is a (connection-oriented) network representation of a fuzzy system $F_{\mathcal{R}}$ in the form of a neural network* $(U, W, A, O, \mathrm{NET}, \mathrm{ex})$ *where*

(i) $U = U_1 \cup U_2 \cup U_3$ *with*
$U_1 = \{x_1, \ldots, x_n\}$, $U_2 = \{R_1, \ldots, R_r\}$, $U_3 = \{y_1, \ldots, y_m\}$.

(ii) W, *the network structure, is a partial mapping from* $U \times U \to \mathcal{F}(\mathbb{R})$ *and is given by*

$$W(u,v) = \begin{cases} \mu_j^{(i)} & \text{if } u = x_i,\ v = R_j \\ \nu_j^{(k)} & \text{if } u = R_j,\ v = y_k \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $1 \le i \le n$, $1 \le j \le r$ *and* $1 \le k \le m$. *In addition every two connections with weights* $W(u,v)$ *and* $W(u',v')$ *become coupled connections, if* $W(u,v) = W(u',v')$ *and* $(u = u',\ u, u' \in U_1 \wedge v \neq v',\ v, v' \in U_2)$ *or* $(u \neq u',\ u, u' \in U_2 \wedge v = v',\ v, v' \in U_3)$ *holds.*

If $W(u,v)$ *and* $W(u',v')$ *are coupled, then if* $W(u,v)$ *is modified by a learning algorithm,* $W(u',v')$ *is modified in the same way and vice versa.*

(iii) A *is a mapping that assigns an activation function* A_u *to each* $u \in U$, *with*

$$A_u \ : \ \mathbb{R} \to \mathbb{R},\ a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u \text{ for } u \in U_1 \cup U_2 \text{ and}$$

$$A_u \ : \ \mathcal{F}(\mathbb{R}) \to \mathcal{F}(\mathbb{R}),\ a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u \text{ for } u \in U_3.$$

(iv) O *is a mapping that assigns an output function* O_u *to each* $u \in U$, *with*

$$O_u \ : \ \mathbb{R} \to \mathbb{R},\ o_u = O_u(a_u) = a_u \text{ for } u \in U_1 \cup U_2 \text{ and}$$

$$O_u \ : \ \mathcal{F}(\mathbb{R}) \to \mathbb{R},\ o_u = O_u(a_u) = \mathrm{defuzz}(a_u) \text{ for } u \in U_3.$$

(v) NET *is a mapping that assigns a network input function* NET_u *to each* $u \in U$, *with*

$$\mathrm{NET}_u \ : \ \mathbb{R} \to \mathbb{R},\ \mathrm{net}_u = \mathrm{NET}_u(\mathrm{ex}_u) = \mathrm{ex}_u \text{ for } u \in U_1,$$

$$\mathrm{NET}_u \ : \ (\mathbb{R} \times \mathcal{F}(\mathbb{R}))^{U_1} \to [0,1],$$

$$\mathrm{net}_u = \top_1 \{W(u',u)(o_{u'})\} \text{ for } u \in U_2 \text{ and}$$

$$\mathrm{NET}_u \ : \ ([0,1] \times \mathcal{F}(\mathbb{R}))^{U_2} \to \mathcal{F}(\mathbb{R}),$$

$$\mathrm{net}_u : \mathbb{R} \to [0,1],$$

$$\mathrm{net}_u(y) = \bot \{\top_2\{o_{u'}, W(u',u)(y)\}\} \text{ for } u \in U_3.$$

(vi) $\mathrm{ex} : U_1 \to \mathbb{R}$, *defines for each input unit* $u \in U_1$ *its external input* $\mathrm{ex}(u) = \mathrm{ex}_u$. *For all other units* ex *is not defined.*

In [Nauck, 1994b, Nauck et al., 1997] we find a definition for a *generic* fuzzy perceptron that is more general than the fuzzy perceptron of Definition 3.12, as it does not enforce coupled connections. A generic fuzzy perceptron can be used to derive neuro-fuzzy models for special domains, and can serve as a common foundation to evaluate different neuro-fuzzy approaches by means of the same underlying model.

In this thesis we are interested in creating an interpretable fuzzy system for data analysis. One important feature of interpretable fuzzy systems is that no linguistic expression is represented by more than one fuzzy set (Section 3.4). Therefore we must take care that a connection-oriented network representation uses coupled connections (shared weights), as required in Definition 3.12.

3.4 Interpretable Fuzzy Systems for Data Analysis

This thesis is about neuro-fuzzy systems in data analysis. As we have seen in the previous chapter, neuro-fuzzy systems are essentially fuzzy systems endowed with learning capabilities inspired by neural networks. We must therefore consider what advantages there are in using fuzzy systems for data analysis?

Fuzzy systems conveniently allow us to model a partially known dependency between independent and dependent variables by using linguistic rules. By using linguistic terms represented by fuzzy sets to describe values, we can select a certain granularity under which the data is observed. We can use a fuzzy system both for predicting values for the dependent variables and for knowledge representation.

Thus a fuzzy system can be regarded as a model that links models for prediction and models for understanding. Usually models for prediction are either not interpretable – they are black boxes like, e.g. neural networks – or they are only interpretable by experts – regression models, for example, offer some interpretation of an underlying process. Models for prediction usually do not need to bother with understandability. Their objective is to create accurate predictions.

Models for understanding are usually represented in some kind of rule base, for example, symbolic rules based on predicate calculus. Such rule bases can help to understand an underlying process but building them is often only feasible for finite categorical domains or if numeric domains are partitioned into crisp sets. Such rule bases are less suitable for prediction because counterintuitive results can occur at the boundaries of sets or domains. This is one of the problems of symbolic expert systems [Dreyfus, 1979].

Fuzzy systems have numeric interpolation capabilities and are therefore well-suited for function approximation and prediction. On the other hand they partition variables by fuzzy sets that can be labeled with linguistic terms. Thus they also have a symbolic nature and can be intuitively interpreted. However, there is a trade-off between readability and precision. We can force fuzzy systems to arbitrary precision, but we then we lose interpretability. To be very precise, a fuzzy system needs a fine granularity and many fuzzy rules. It is obvious that the larger the rule base of a fuzzy system becomes the less interpretable it gets.

If we are interested in a very precise prediction, then we are usually not so much interested in the interpretability of the solution. In this case we want to use another feature of fuzzy systems: the convenient combination of local models to an overall solution. For this, Sugeno-type models are more suited than Mamdani-type models because they offer more flexibility in the consequents of the rules [Grauel and Mackenberg, 1997]. However, if we need a precise model for prediction we should consider whether a fuzzy system is the most suitable approach. Very similar to Sugeno-type fuzzy systems are radial basis function networks, kernel regression or B-spline networks [Bersini and Bontempi, 1997a, Brown and Harris, 1994].

If we are more interested in an interpretable solution, then a Mamdani-type fuzzy system should be preferred, because the rule consequents consist of interpretable fuzzy sets. If the fuzzy system is generated in a data analysis process by a neuro-fuzzy learning procedure, then we must take into account that we will probably not obtain a very precise solution. We cannot allow a learning algorithm to apply all the possible modifications to the parameters of a fuzzy systems. For the sake of interpretability we must constrain the learning procedure (see also Section 5.3).

In the area of data analysis the interpretability and simplicity of fuzzy systems are the key advantage. Fuzzy systems are not better function approximators or classifiers than other approaches. If we want to keep the model simple, the prediction is usually less acurate. This means fuzzy systems should be used for data analysis, if an interpretable model is needed that can also be used to some extent for prediction. The interpretability of fuzzy systems is discussed, for example, in [Bersini and Bontempi, 1997b, Bersini et al., 1998, Nauck and Kruse, 1997d, Nauck and Kruse, 1998c, Nauck and Kruse, 1999b].

Interpretability of a model is especially important in areas

- where humans usually make decisions, but where machines can now support the decision making process or even take full responsibility for it,
- where prior knowledge is to be used in the data analysis process and the modification of this knowledge by a learning process must be checked,
- where solutions must be explained or justified to non-experts.

Interpretability of a fuzzy model should not mean that there is an exact match between the linguistic description of the model and the model parameters. This is not possible anyway, due to the subjective nature of fuzzy sets and linguistic terms. Usually it is not important that, for example, the term *approximately zero* be represented by a symmetrical triangular fuzzy set with support $[-1, 1]$. Interpretability means that the users of the model can accept the representation of the linguistic terms, more or less. The representation must roughly correspond to their intuitive understanding of the linguistic terms. It is more important that the rule base is small and thus comprehensible. It is also useful to note that interpretability itself is a fuzzy and subjective concept.

Furthermore, interpretability should not mean that *anybody* can understand a fuzzy system. It means that users who are at least to some degree experts in the domain where the data analysis takes place can understand the model. Obviously we cannot expect a lay person to understand a fuzzy system in a medical domain. It is important that the medical expert who uses the model should understand it.

It can also be useful to distinguish between *local interpretability* and *global interpretability* (Table 3.1). Usually we are interested in globally interpretable models, i.e. models that can be understood as a whole and that provide an overview. But sometimes, especially in technical processes, local interpretability may be sufficient. In these cases there is knowledge about the local behaviour of the underlying process and it is more important to accurately represent these local aspects. An overview about the complete model is missing in these cases. Fuzzy controllers, for example, are often locally interpretable [Babuska, 1998].

Table 3.1: Global and local interpretability

Global Interpretability	Local Interpretability
the model can be understood as a whole	only local aspects of the model can be understood, the overview is missing
coarse model	fine model
few parameters	many parameters
low precision, i.e. many errors = high costs in application	high precision, i.e. few errors = low cost in application
low costs in model creation	high costs in model creation

From the viewpoint of a user we can formulate the following intuitive criterion for the interpretability of a fuzzy system. We assume that the linguistic interpretability of a fuzzy system is adequate if

- it provides a rough idea about the underlying process or the relations within the data,
- it sufficiently justifies the majority of observed output values,
- it is usable for explanations,
- it covers all important observed input/output situations (rare cases or exceptions might be ignored).

A neuro-fuzzy learning procedure for creating interpretable fuzzy systems in data analysis must be simple and fast to allow a user to understand what it does and to experiment with it. We prefer a tool-oriented, exploratory view on neuro-fuzzy systems. We consider a neuro–fuzzy method to be a tool for creating fuzzy systems from data. The learning algorithm should take the semantics of the desired fuzzy system into account, and adhere to certain constraints. The learning result should also be interpreted, and the insights gained by this should be used to restart the learning procedure to obtain better results if necessary. A neuro-fuzzy system supports the user in finding a desired fuzzy system based on training data, but it cannot do all the work. This view matches the exploratory nature of intelligent data analysis (Section 2.1).

Semantical problems will occur if neuro-fuzzy systems do not have mechanisms to make sure that all changes caused by the learning procedure are interpretable in terms of a fuzzy system. The learning algorithms should be constrained such that adjacent membership functions do not exchange positions, do not move from positive to negative parts of the domains or vice versa, have a certain degree of overlapping, etc. An interpretation in terms of a Mamdani-type fuzzy system may not be possible if the evaluation of antecedents is not done by t-norms, but by certain special functions. Sometimes this is done to allow the application of gradient descent learning.

The following points influence the interpretability of a fuzzy system:

- The number of fuzzy rules: a fuzzy system with a large rule base is less interpretable than a fuzzy system that needs only few rules.
- The number of variables: high dimensional models are incomprehensible. Each rule should use as few variables as possible.
- The number of fuzzy sets per variable: only a few meaningful fuzzy sets should be used to partition a variable. A fine granularity not only increases the number of linguistic terms for a variable, but also the number of possible fuzzy rules increases exponentially with the number of variables and fuzzy sets. A coarse granularity increases the readability of the fuzzy model.

- Unambiguous representation of linguistic terms: each linguistic term must be represented by only one fuzzy set. Different rules using the same linguistic expression (e.g. x is *small*) may not represent the corresponding linguistic term (e.g. *small*) by different fuzzy sets.

- No conflicts: there must be no rules in the rule base that have identical antecedents but different consequents (complete contradiction). Only partial contradiction is acceptable.

- No redundancy: no rule may appear more than once in the rule base. There must also be no rule whose antecedent is a subset of the antecedent of another rule.

- Characteristics of fuzzy sets: fuzzy sets should be "meaningful" to the user of the fuzzy system. After training, the fuzzy partition of a variable should still be reasonably similar to the partition provided by the user. At least the relative position of the fuzzy sets must be maintained. Usually, a minimum/maximum degree of overlapping must be enforced. Fuzzy sets should be normal and convex and be interpretable as fuzzy numbers or fuzzy intervals (for numeric variables, symbolic variables are discussed in Section 4.3).

The points given above must be observed by neuro-fuzzy learning techniques. In Chapters 4 and 5 we will discuss these points together with the learning algorithms presented there. The characteristics of the fuzzy sets are enforced by constraints (Section 5.3). The number of parameters (rules, variables) can be decreased after learning by pruning methods (Section 5.5). Fuzzy decision tree learning methods try to use as few variables as possible from the beginning. The granularity of the fuzzy system can only be determined by learning, if methods from cluster analysis are used (Section 4.1). However, these methods have other problems connected with interpretability as we shall see. Conflicts, redundany and ambiguity can be avoided by proper implementation of the fuzzy system and the learning algorithm.

Chapter 4

Learning Fuzzy Rules from Data

The structure of a fuzzy system is given by its rules and by the granularity of the data space, i.e. the number of fuzzy sets used to partition each variable. The parameters of a fuzzy system are the shapes and locations of the membership functions.

Assume we want to determine whether there is a fuzzy system that yields an error value below a given threshold ε for a particular learning problem. To do this we can enumerate rule bases until we find such a fuzzy system or until all possible rule bases are checked.

In order to restrict the number of possible rule bases, we examine a simplified scenario where we consider Mamdani-type fuzzy systems that use q triangular fuzzy sets for each variable. We assume that for each variable $x \in [l, u] \subset \mathbb{R},\ l < u$, holds. A membership function $\mu_{a,b,c}$ is given by three parameters $a < b < c$. Instead of selecting parameters $a, b, c \in \mathbb{R}$, which would result in an infinte number of possible membership functions, we sample the variables such that there are $m + 2$ samples $l = x_0 < x_1 < \ldots < x_m < x_{m+1} = u$ for each variable x. We assume $m \geq q$. The jth fuzzy set ($j \in \{1, \ldots, q\}$) of x is given by $\mu_{x_{k_{j-1}}, x_{k_j}, x_{k_{j+1}}},\ k_j \in \{1, \ldots, m\}$, $k_{j-1} < k_j < k_{j+1}$. We define $k_0 = 0$ and $k_{q+1} = m + 1$. Thus we obtain for each variable x a fuzzy partition, where the degrees of membership add up to one for each value of x. There are $\binom{m}{q}$ possible fuzzy partitions for each variable.

A fuzzy set configuration is given by the fuzzy partitions for all variables. If there are n variables, then we can choose between $\binom{m}{q}^n$ fuzzy set configurations. For each configuration there are $(q + 1)^n$ possible rules, because a rule can either include a variable by selecting one of its q fuzzy sets or the variable is not used by the rule.

A rule base can be any subset of all possible rules, i.e. there are $2^{((q+1)^n)}$ possible fuzzy rule bases for each fuzzy set configuration. Altogether, there are

$$\binom{m}{q}^n 2^{((q+1)^n)}$$

possible fuzzy rule bases.

These considerations show that finding an appropriate fuzzy system by simply enumerating fuzzy rule bases becomes intractable even for moderate values of n, q and m. Thus there is a need for data driven heuristics to create fuzzy systems. In this and in the following chapter we discuss such methods.

Both structure and parameters can be derived from training data. Before the parameters of a fuzzy system can be optimized in a training process the structure – the rule base – must be determined. One benefit of fuzzy systems is that the rule base can be created from expert knowledge. However, in many applications expert knowledge is only partially available or not at all. In these cases it must be possible to create a rule base from scratch relying only on the training data.

If the granularity and the rules are to be determined at the same time, then unsupervised learning like cluster analysis can be used, as described in Section 4.1. These approaches have drawbacks when the resulting rule base must be interpretable. Readability of the solution can be guaranteed more easily, if the granularity of the data space is defined in advance and the data space is structured by pre-defined fuzzy partitions for all variables. Section 4.2 describes supervised rule learning algorithms that are based on a structured data space.

Reinforcement learning [Kaelbling et al., 1996] is a special form of supervised learning that can be used for rule learning in fuzzy control applications. Such algorithms are not considered, because problems where reinforcement signals are used instead of unknown output values are not relevant for data analysis problems. For an overview on reinforcement learning in fuzzy systems see [Nauck, 1994b, Nauck et al., 1997]. Recent advances can be found in [Nürnberger et al., 1999].

When real world data sets must be analysed we often have to deal with different types of variables on different scales, i.e. nominal scales (categorical or symbolic data), ordinal scales, or interval and ratio scales (both metric). Data analysis approaches often have problems handling symbolic and numeric data at the same time. The most common approach is to either represent symbols numerically or to use intervals for numeric data.

For example, neural networks, many statistical procedures like regression and cluster analysis or pattern analysis methods rely on metric data. To process nominal scaled data these approaches usually represent them on artificial metric scales. This can lead to undesired results, because the approaches interpret distances and ratios between values, which are actually meaningless for numerically represented symbols.

Approaches like decision trees [Quinlan, 1993], Bayesian networks [Kruse et al., 1991, Pearl, 1988] or logic-based approaches work best with symbolic data, or at least discrete finite domains. To deal with continuous variables, they must use intervals. This approach can become computationally expensive, if many intervals

must be used, or counterintuitive results can be produced due to improperly selected interval boundaries.

Fuzzy systems are not dependent on the scales of the data they process. One of their biggest benefits for data analysis is that they can provide solutions that are interpretable in terms of the involved variables. It would therefore be useful to be able to create fuzzy rules from data that contain symbolic variables without representing them numerically. In Section 4.3 a rule learning algorithm for such a scenario is described.

In processing real world data we often must deal with missing values. Many neuro-fuzzy learning algorithms cannot cope with this problem and simply delete incomplete patterns from the learning problem. This, however, can lead to a substantial or even inacceptable loss of training data. An approach to learning fuzzy rules if the data contain missing values is described in Section 4.4. The last section of this chapter anlyzes the complexity of the discussed learning algorithms.

4.1 Structure Learning

Before a fuzzy system can be optimized in a training process, its structure must be defined, i.e. a fuzzy rule base must be created. In this Section we discuss three general approaches: *cluster-oriented*, *hyperbox-oriented* and *structure-oriented* fuzzy rule learning. The first two approaches create fuzzy rules and fuzzy sets at the same time. The third approach needs initial fuzzy partitions for all variables to create a rule base.

Cluster-oriented and Hyperbox-oriented Fuzzy Rule Learning

Cluster-oriented methods try to group the training data into clusters and use them to create rules. Fuzzy cluster analysis [Bezdek, 1981, Bezdek et al., 1998] can be used for this task by searching for spherical or hyperellipsoidal clusters. The clusters are multidimensional (discrete) fuzzy sets which overlap. An overview on several fuzzy clustering algorithms can be found, for example, in [Höppner et al., 1999].

Each fuzzy cluster can be transformed into a fuzzy rule, by projecting the degrees of membership of the training data to the single dimensions. Thus for each cluster and each variable a histogram is obtained that must be approximated either by connecting the degrees of membership by a line, by a convex fuzzy set, or – more preferably – by a parameterized membership function that should be both normal and convex and fits the projected degrees of memberships as well as possible [Klawonn and

Kruse, 1995, Sugeno and Yasukawa, 1993]. This approach can result in forms of membership functions which are difficult to interpret (see Figure 4.1)

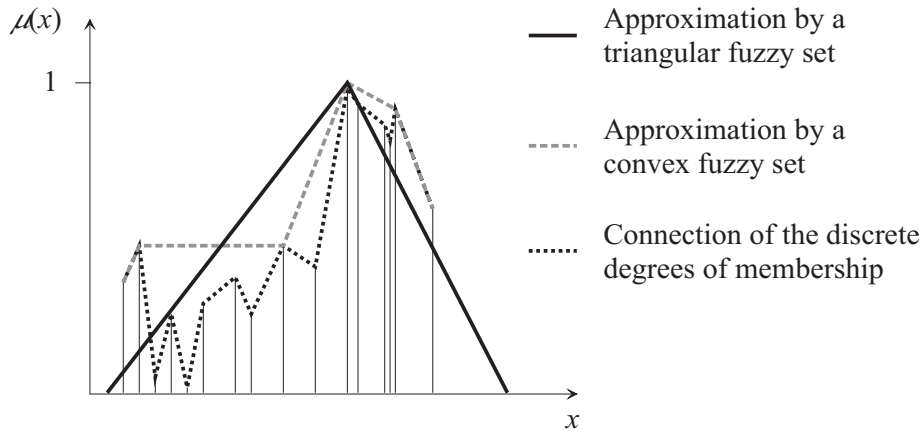


Figure 4.1: Creation of a fuzzy set from projected degrees of membership

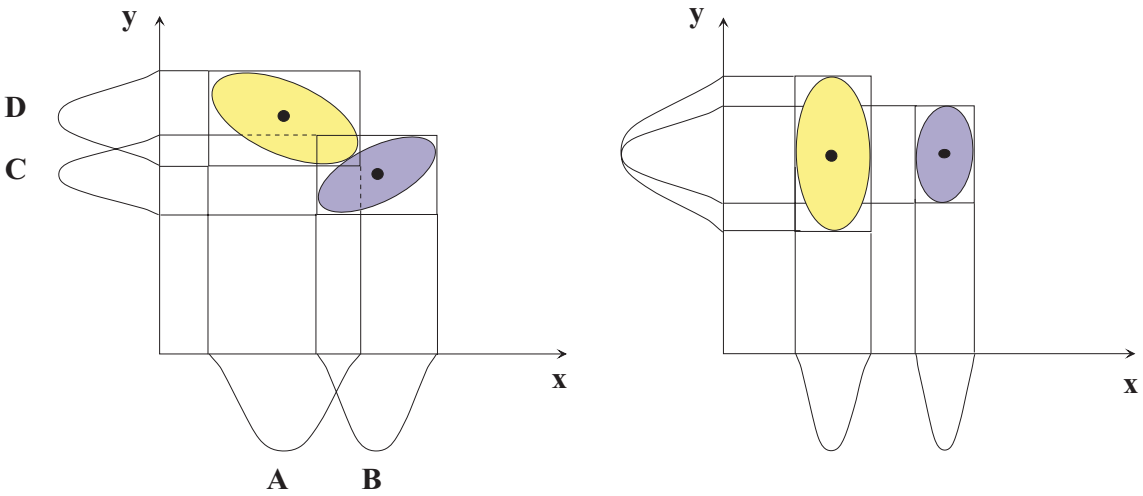


Figure 4.2: If clusters in the form of hyperellipsoids are projected to obtain fuzzy rules, a loss of information occurs and unusual fuzzy partitions can be obtained

This procedure also causes a loss of information because the Cartesian product of the induced membership functions does not reproduce a fuzzy cluster exactly (see Figure 4.2). This loss of information is strongest in the case of arbitrarily oriented hyperellipsoids. To make this problem easier to handle, it is possible to search for axes–parallel hyperellipsoids only [Klawonn and Kruse, 1997].

The fuzzy rule base obtained by projecting the clusters is usually not easy to interpret, because the fuzzy sets are induced individually for each rule (Figure 4.2). For

each feature there will be as many different fuzzy sets as there are clusters. Some of these fuzzy sets may be similar, yet they are usually not identical. For a good interpretation it is necessary to have a fuzzy partition of few fuzzy sets where each clearly represents a linguistic concept (see discussion in Section 3.4).

The loss of information that occurs in projecting fuzzy clusters can be avoided, if the clusters are hyperboxes and parameterized multidimensional membership functions are used to represent a cluster. As in fuzzy cluster analysis the clusters (hyperboxes) are multidimensional overlapping fuzzy sets. The degree of membership is usually computed in such a way that the projections of the hyperboxes on the individual variables are triangular or trapezoidal membership functions (Figure 4.3)

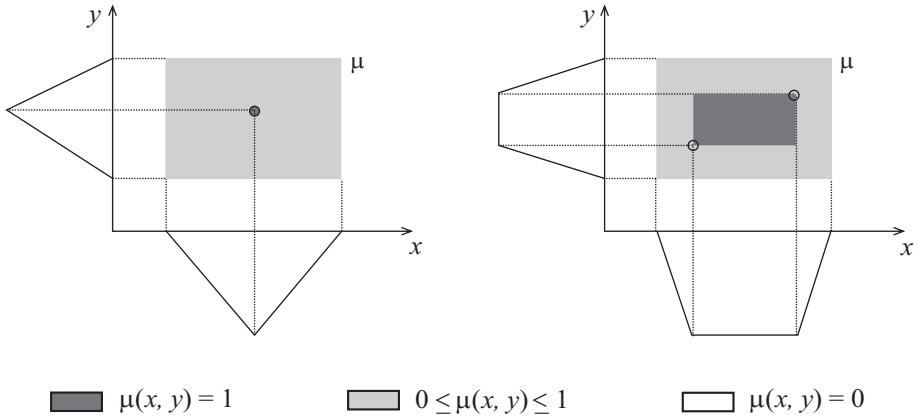


Figure 4.3: Hyperboxes as multidimensional fuzzy sets

Hyperbox-oriented fuzzy rule learning is usually supervised. For each pattern of the training set that is not covered by a hyperbox, a new hyperbox is created and the output of the training pattern (class information or output value) is attached to this new hyperbox. If a pattern is incorrectly covered by a hyperbox with different output, this hyperbox is shrunk. If a pattern is correctly covered by a hyperbox it is enlarged to increase the degree of membership for the pattern. Figure 4.4 demonstrates this approach for the XOR problem.

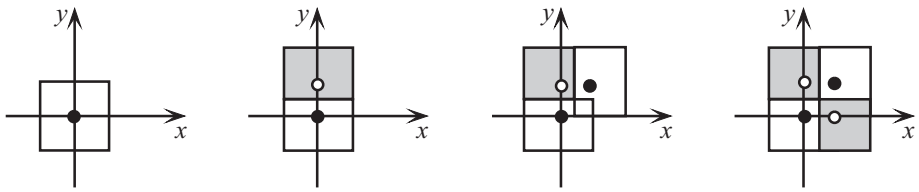


Figure 4.4: The XOR problem solved by creating four hyperboxes

Like in fuzzy cluster analysis fuzzy rules can be created by projecting the hyperboxes (see Figure 4.5). Thus the learning algorithm creates a rule base and membership

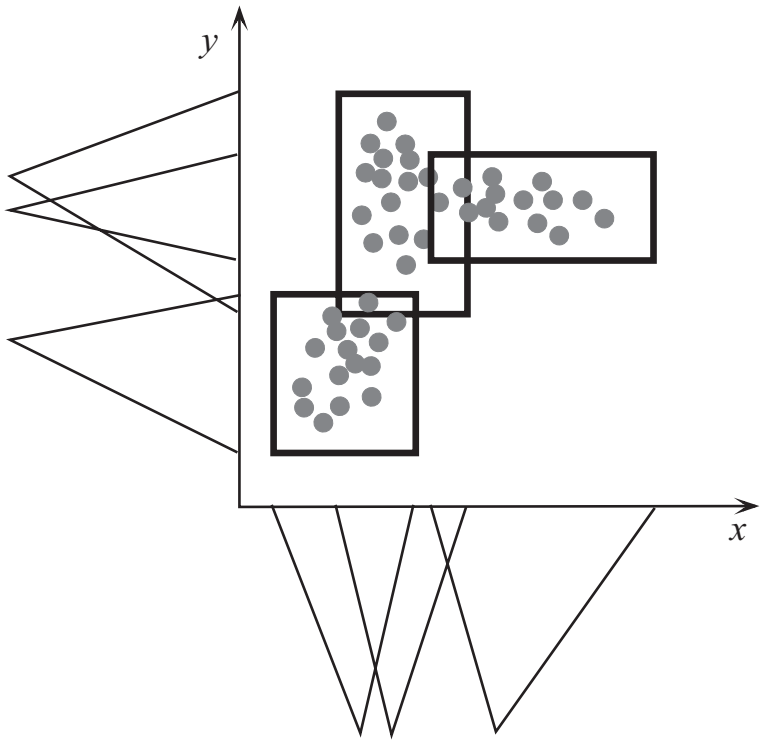


Figure 4.5: Searching for hyperboxes to create fuzzy rules and fuzzy sets

functions at the same time. However, each fuzzy rule creates its own fuzzy sets for each variable and this usually results in rule bases that cannot be interpreted very well.

Hyperbox-oriented fuzzy rule learning is computationally less demanding than fuzzy cluster analysis and can create solutions for benchmark problems in pattern recognition or function approximation very fast [Berthold and Huber, 1999, Tschichold Gürman, 1996]. If there are no contradictions in the training patterns and if there is only one output variable, then hyperbox-oriented learning algorithms can create solutions with no errors on the training data. In the worst case this leads to a situation, where each training pattern is covered by its indvidual hyperbox.

Grid clustering [Klawonn and Keller, 1997, Keller and Klawonn, 1998] is a method that can be viewed as a combination of several fuzzy rule learning methods. Each domain is partitioned by fuzzy sets. Then an unsupervised learning algorithm modifies the fuzzy sets to improve the partitioning of the data space. It is required that for each pattern the degrees of membership add up to 1. This approach can be viewed as a cluster anlysis, where the clusters are hyperboxes which are aligned on a grid. It can therefore also be interpreted as hyperbox-oriented. Because the data space is structured by predefined fuzzy sets it can also be viewed as a structure-oriented learning method as discussed in Section 4.2. The clusters are given by membership functions and not vice versa. Therefore the rule base obtained by grid

clustering can be well interpreted, because the fuzzy rules do not use individual fuzzy sets.

It is also possible to use neural networks to create fuzzy rule bases. RBF networks (Definition 3.10) can be used to obtain a (Sugeno-type) fuzzy rule base. An RBF network uses multi-dimensional radial basis functions in the nodes of its hidden layer. Each of these functions can be interpreted as a fuzzy cluster. If the RBF network is trained by a gradient descent procedure it adjusts the location of the radial basis functions and – depending on the type of network – also their size and orientation. A fuzzy rule base is determined after training by projecting the radial basis functions.

Pedrycz and Card suggested a way to linguistically interpret Kohonen's self-organizing feature maps (SOM) [Kohonen, 1984] in order to create a fuzzy rule base [Pedrycz and Card, 1992]. A feature map is used to perform a cluster analysis on the training data and thus to reduce the data set to a set of prototypes represented by neurons. The prototypes are then used to create a fuzzy rule base by a structure-oriented approach as they are discussed in Section 4.2.

Kosko suggested a cluster-oriented approach to fuzzy rule learning that is based on his FAM model (Fuzzy Associative Memory) [Kosko, 1992]. Kosko uses a form of adaptive vector quantization that is not topology preserving as in the case of a SOM. In addition, Kosko's procedure determines weights for the resulting fuzzy rules. The problems of using weighted rules is discussed in Section 5.1

All the approaches that are discussed above have drawbacks when interpretable fuzzy systems for data analysis must be created. Cluster-oriented and hyperbox-oriented approaches to fuzzy rule learning both have the following problems:

- each fuzzy rule uses individual fuzzy sets,
- fuzzy sets obtained by projection are hard to interpret linguistically,
- they can only be used for metric data, and
- they cannot cope with missing values.

Cluster-oriented approaches also have the following restrictions:

- they are unsupervised and do not optimize an output error or performance measure,
- a suitable number of clusters must be determined by repeating the cluster analysis with an increasing number of clusters until some validity measure assumes a local optimum,
- the algorithms can become computationally very expensive, especially if clusters are arbitrarily rotated hyperellipsoids and

- a loss of information occurs, if fuzzy rules are created by projection.

Grid clustering does not have most of the drawbacks of fuzzy cluster analysis, and it creates interpretable fuzzy rule bases. However, it is an unsupervised method and therefore its application is restricted.

Fuzzy cluster analysis is very well suited in segmentation tasks [Grauel et al., 1997, Höppner et al., 1999] – especially in areas where linguistic interpretation plays a minor role. In data analysis fuzzy cluster analysis can be helpful during preprocessing. The cluster analysis can reveal how many rules are needed to suitably partition the data space and can give some insights into the data. It is possible to map the fuzzy sets obtained by clustering to previously defined fuzzy sets [Klawonn et al., 1995b, Nauck and Klawonn, 1996] and thus obtain an initial rule base that can be tuned further by training the membership functions (see Chapter 5).

Because the main objective of the data analysis approaches discussed in this thesis is to create interpretable fuzzy rule bases, cluster-oriented and hyperbox-oriented approaches to fuzzy rule learning are considered to be less useful in this context.

Structure-oriented Fuzzy Rule Learning

Structure-oriented approaches can be seen as special cases of hyperbox approaches that do not search for clusters in the data space, but select hyperboxes from a grid structure. By providing (initial) fuzzy sets for each variable the data space is structured by overlapping hyperboxes (compare Figure 4.6). This way of learning fuzzy rules was suggested by Wang and Mendel [Wang and Mendel, 1991, Wang and Mendel, 1992].

To apply the Wang&Mendel algorithm all variables are partitioned by fuzzy sets. For this purpose equidistant overlapping triangular or trapezoidal membership functions are usually used. By this means the feature space is partitioned by overlapping multidimensional fuzzy sets whose support is a hyperbox. Rules are created by selecting those hyperboxes that contain data. Wang and Mendel designed their algorithm to create fuzzy systems for function approximation. In order to mediate between different output values for the same combination of input values, they used weighted rules. In [Wang and Mendel, 1992] a proof can be found that this algorithm can create fuzzy rule bases that can approximate any real continuous function over a compact set to an arbitrary accuracy.

In [Higgins and Goodman, 1993] a variation of the Wang&Mendel algorithm was suggested: this creates fuzzy partitions during rule creation by refining the existing partitions. The algorithm begins with only one membership function for each variable, such that the whole feature space is covered by one large hyperbox. Subsequently, new membership functions are inserted at points of maximum error by refining the fuzzy partitions of all variables. Then the old rules are discarded and

a new set of rules is generated based on the new fuzzy partitions. This procedure is iterated until a maximum number of fuzzy sets is created or the error decreases below some threshold. This algorithm was created in order to compensate for a drawback of the Wang&Mendel algorithm, which has problems modeling extreme values of the function to be approximated. However, the Higgins&Goodman algorithm tends to fit outliers because it concentrates on areas with large error.

Fuzzy decision trees are another approach to structure-oriented fuzzy rule learning. Induction of decision trees [Quinlan, 1986, Quinlan, 1993] is a very popular approach in data analysis to generate classification or regression models. Decision trees are based on discriminative learning algorithms working by means of recursive partitioning. The data space is partitioned in a data-driven manner and the partition is represented as a tree. A decision tree can be transformed into a rule base, by following each path from the root node to a leaf node. Well-known algorithms are, for example, ID3 [Quinlan, 1986] for symbolic domains and C4.5 [Quinlan, 1993] that can also incorporate numeric variables. Both approaches are applied to classification problems. CART [Breiman et al., 1984] is an approach that can create decision trees for classification and regression.

Algorithms for building decision trees at the same time try to optimize the performance of the tree and to create a tree as small as possible. This is done by selecting attributes to be included into the tree according to some information theoretical measure like, for example, information gain [Quinlan, 1993]. A comparative study on different selection measures can be found in [Borgelt and Kruse, 1998].

Fuzzy decision trees [Boyen and Wehenkel, 1999, Ichihashi et al., 1996, Janikow, 1996, Janikow, 1998, Yuan and Shaw, 1995] extend the idea of decision tree learning to the domain of fuzzy systems. Instead of propagating a pattern through the tree based on crisp tests on attribute values, fuzzy tests are used. Each variable of the considered problem must be previously partitioned by fuzzy sets. A test of an attribute in a fuzzy decision tree means determining the degree of membership of the attribute value to a fuzzy set.

Fuzzy decision trees can be viewed as a structure-oriented rule learning procedure with concurrent structure optimization. By selecting attributes with high information content first, it may turn out that not all variables are needed to solve the learning problem. However, this approach is heuristic and there is no guarantee that the induced tree is optimal in some sense – either in structure (size) or in performance.

The advantage of fuzzy decision tree learning is that not all variables must be included in the rule base at once, as is the case for other structure-oriented or cluster-oriented rule learning approaches which becomes difficult for high-dimensional learning problems. By restricting the height of the tree small rule bases can be enforced even for high-dimensional problems, if a possible loss of performance can be tolerated.

Because the induction of a decision tree is based on heuristics, it can happen that rule learning procedures that include all variables at once produce better results with the drawback of a large rule base. But if a pruning algorithm is applied to such a large rule base it is often possible to reduce the rule base and to retain the performance. In the following we will therefore concentrate on structure-oriented learning algorithms that use all variables in the beginning. After training, the created fuzzy systems are optimized by pruning methods as discussed in Section 5.5).

Compared to fuzzy clustering or hyperbox-oriented approaches, structure-oriented approaches to fuzzy rule generation have the following advantages:

- they can create rule bases that can easily be interpreted linguistically
- they are very fast and computationally inexpensive,
- they are very easy to implement,
- they can be used if the data contains numeric and non-numeric attributes,
- they can be used if the data contains missing values.

Structure-oriented approaches to fuzzy rule learning are therefore more suitable in data analysis. In the following section we present algorithms to create Mamdani-type fuzzy rule bases for function approximation and classification. Afterwards we show how these algorithms can be extended to handle non-numeric attributes and missing values.

4.2 Learning Mamdani-type Fuzzy Rules

The algorithms presented in this section are extensions to the approach by Wang & Mendel [Wang and Mendel, 1992] and are used by the neuro-fuzzy approaches NEFCLASS [Nauck and Kruse, 1995, Nauck and Kruse, 1997b, Nauck and Kruse, 1998b] and NEFPROX [Nauck and Kruse, 1997a, Nauck and Kruse, 1998c, Nauck and Kruse, 1999a]. NEFCLASS is used for classification problems and NEFPROX for function approximation. The rule learning algorithms for both approaches refrain from using rule weights and determine the best consequent for a rule by a performance measure or by using an average value. In addition the algorithm tries to reduce the size of the rule base by selecting only a number of rules depending on their performance or on the coverage of the training data. This algorithm is now described in detail.

All variables of the considered problem must be partitioned by fuzzy sets before rule learning can take place. If a domain expert or a user provides these fuzzy sets and labels them appropriately, then these labels can be taken as the “vocabulary”

used to describe the problem solution represented by the rule base to be learned. If the fuzzy sets are selected such that they are meaningful to the user, then the interpretability of the rule base depends only on the number of rules and variables (compare Section 3.4). The fuzzy sets can be selected individually for each variable in order to model individual granularities under which the variables are to be observed. Thus a user can hide unwanted information or focus on important areas of the domains of the variables.

Even if a user does not want to individually specify fuzzy sets and prefers to simply use fuzzy partitions of equidistant overlapping membership functions like triangular, trapezoidal or bell-shaped functions, the interpretability of the created rule base will be high. Such fuzzy sets can be conveniently interpreted as fuzzy numbers or fuzzy intervals. Besides, there will be no individual fuzzy sets for each rule as in cluster-oriented or hyperbox-oriented rule learning methods. All created rules share the same fuzzy sets. Thus it is not possible that a linguistic value is represented by different fuzzy sets in a rule base.

The number of membership functions also defines the granularity of the data space. If there are n (input + output) variables, then each hyperbox is a Cartesian product of n fuzzy sets, one from each variable. The number of hyperboxes is equal to the number of all possible fuzzy rules that can be created using the given fuzzy sets.

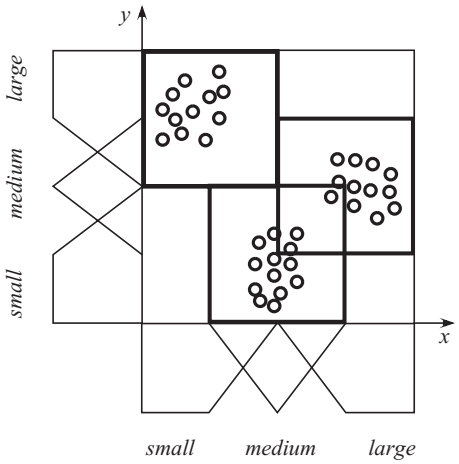


Figure 4.6: Structure-oriented approaches use initially defined fuzzy sets to structure the data space by overlapping hyperboxes, which represent fuzzy rules

From an implementation point of view there is no definite requirement to actually create all these hyperboxes (fuzzy rules) and store them in the memory of a learning algorithm. The rules are created on the fly by processing the training data set twice. In the beginning, the rule base is either empty, or contains some rules provided as prior knowledge. In the first cycle all the required antecedents are created. For each

point of the data set that is used for rule creation a combination of fuzzy sets is selected. This is done by finding, for each variable, the membership function that yields the highest degree of membership for the current input value. If an antecedent combined from those fuzzy sets does not yet exist in the list of antecedents, it is simply added to it. In the second cycle the best consequent for each antecedent is determined and the rules are completed. The maximum number of rules is bound from above by

$$\min\left\{s,\ \prod_{i}^{n} q_i\right\}$$

where s is the cardinality of the training data set and q_i is the number of fuzzy sets provided for variable x_i. If the training data has a clustered structure and concentrates only in some areas of the data space, then the number of rules will be much smaller then the theoretically possible number of rules. The actual number of rules will normally be bound by criteria defined by the user like "create no more than k rules" or "create so many rules that at least $p\%$ of all training data are covered".

The suitability of the rule base depends on the initial fuzzy partitions. If there are too few fuzzy sets, groups of data that should be represented by different rules might be covered by a single rule only. If there are more fuzzy sets than necessary to distinguish different groups of data, too many rules will be created and the interpretability of the rule base decreases. The example in Figure 4.6 shows three clusters of data that are represented by the following three rules:

if x is *small*	then y is *large*
if x is *medium*	then y is *small*
if x is *large*	then y is *medium*

In Algorithms 4.1 – 4.4 we present procedures for structure-oriented fuzzy rule learning in classification or function approximation problems. The algorithms are implemented in the neuro-fuzzy approaches NEFCLASS and NEFPROX. Depending on the problem the consequent of a rule is either a a class label or a fuzzy set. Here we consider Mamdani-type algorithms only. However, the algorithms can be extended easily to Sugeno-type fuzzy rules by providing suitable initial linear models as consequents for the rules. The coefficients of the linear models can be determined, for example, by the ANFIS algorithm afterwards (see Section 5.2).

We begin with the NEFCLASS rule learning algorithm (Algorithms 4.1 – 4.3) which uses the following notations:

- $\tilde{\mathcal{L}}$: a set of training data (fixed learning problem) with $\left|\tilde{\mathcal{L}}\right| = s$, which represents a classification problem where patterns $\mathbf{p} \in \mathbb{R}^n$ are to be assigned to m classes $C_1, \ldots, C_m$, with $C_i \subseteq \mathbb{R}^n$.

- $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$: a training pattern consists of an input vector $\mathbf{p} \in \mathbb{R}^n$ and a target vector $\mathbf{t} \in [0,1]^m$. The target vector represents a possibly vague classification of the input pattern $\mathbf{p}$. The class index of $\mathbf{p}$ is given by the index of the largest component of $\mathbf{t}$: $\text{class}(\mathbf{p}) = \text{argmax}_j\{t_j\}$.

- $R = (A, C)$: a fuzzy classification rule with antecedent $\text{ant}(R) = A$ and consequent $\text{con}(R) = C$, where $A = (\mu_{j_1}^{(1)}, \ldots, \mu_{j_n}^{(n)})$ and C is a class. We use both $R(\mathbf{p})$ and $A(\mathbf{p})$ to denote the degree of fulfilment of rule R (with antecedent A) for pattern $\mathbf{p}$, i.e. $R(\mathbf{p}) = A(\mathbf{p}) = \min\{\mu_{j_1}^{(1)}(p_1), \ldots, \mu_{j_n}^{(n)}(p_n)\}$.

- $\mu_j^{(i)}$: jth fuzzy set of the fuzzy partition of input variable x_i. There are q_i fuzzy sets for variable x_i.

- $\mathbf{c}_A$: a vector with m entries to represent the accumulated degrees of membership to each class for all patterns with $A(\mathbf{p}) > 0$; $\mathbf{c}_A[j]$ is the jth entry of $\mathbf{c}_A$.

- $P_R \in [-1, 1]$: a value representing the performance of rule R:

$$P_R = \frac{1}{s} \sum_{(\mathbf{p},\mathbf{t}) \in \tilde{\mathcal{L}}} (-1)^c R(\mathbf{p}), \text{ with } c = \begin{cases} 0 & \text{if class}(\mathbf{p}) = \text{con}(R), \\ 1 & \text{otherwise.} \end{cases} \tag{4.1}$$

At first, the rule learning algorithm detects all rule antecedents that cover some training data and creates a list of antecedents. In the beginning this list is either empty, or it contains antecedents from rules given as prior knowledge. Each time an input training pattern is not already covered by an antecedent from the list, a new antecedent is created and stored (Algorithm 4.1). Next, the algorithm selects an appropriate consequent for each antecedent A and creates a list of rule base candidates. For each antecedent, that specific class is selected that accumulated the largest value in the antecedent's vector $\mathbf{c}_A$. A performance measure $P \in [-1, 1]$ is computed for each rule indicating its unambiguity. For $P = 1$ a rule is general and classifies all training patterns correctly. For $P = -1$ a rule classifies all training patterns incorrectly. For $P = 0$ either misclassifications and correct classifications of a rule are more or less equal, or the rule covers no patterns at all. Only rules with $P > 0$ are considered to be useful.

The last part of the learning procedure is given by Algorithms 4.2 and 4.3. They select the final rule base from the list of rule base candidates computed by Algorithm 4.1. The number of rules is determined by one of the following two criteria:

(i) The size of the rule base is bound by $k_{\max}$, a value that is specified by the user.

(ii) The size of the rule base is chosen such that each training pattern is covered by at least one rule.

Algorithm 4.1: The NEFCLASS rule learning algorithm

1: **for all** patterns $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$ **do** *(* there are s training patterns *)*
2: **for all** input features x_i **do**
3: $\mu_{j_i}^{(i)} = \underset{\mu_j^{(i)}, j \in \{1,\ldots,q_i\}}{\operatorname{argmax}} \{\mu_j^{(i)}(p_i)\}$;
4: **end for**
5: Create antecedent $A = (\mu_{j_1}^{(1)}, \ldots, \mu_{j_n}^{(n)})$;
6: **if** ($A \notin$ list of antecedents) **then**
7: add antecedent A to list of antecedents;
8: **end if**
9: **end for**
10: **for all** patterns $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$ **do** *(* sum up degrees of fulfilments *)*
11: **for all** $A \in$ list of antecedents **do** *(* of antecedents for each class *)*
12: $\mathbf{c}_A[\text{class}(\mathbf{p})] = \mathbf{c}_A[\text{class}(\mathbf{p})] + A(\mathbf{p})$;
13: **end for**
14: **end for**
15: **for all** A in list of antecedents **do**
16: $j = \underset{i \in \{1,\ldots,m\}}{\operatorname{argmax}} \{\mathbf{c}_A[i]\}$;
17: create rule R with antecedent A and consequent C_j;
18: add R to list of rule base candidates;
19: $P_R = \frac{1}{s}(\mathbf{c}_A[j] - \sum_{i \in \{1,\ldots,m\},\, i \neq j} \mathbf{c}_A[i])$; *(* performance of rule R *)*
20: **end for**
21:
22: **if** (select best rules) **then**
23: SelectBestRules; *(* see Algorithm 4.2 *)*
24: **else if** (select best rules per class) **then**
25: SelectBestRulesPerClass; *(* see Algorithm 4.3 *)*
26: **end if**

The learning procedure provides two evaluation procedures:

(i) "Best rules": the best rules are selected based on the performance measure such that the criterion for the rule base size is fulfilled (Algorithm 4.2). In this case it may happen, that some classes are not represented in the rule base, if the rules for these classes have low performance values.

(ii) "Best rules per class": for each of the m classes the next best rule is selected alternately until the criterion for the rule base size is fulfilled (Algorithm 4.3). This usually results in the same number of rules for each class. However, this may not be the case, if there are only few rules for some of the classes, or if many rules of some of the classes are needed to fulfil the second rule base size criterion (cover all patterns).

If the rule learning procedure presented in Algorithm 4.1 is to be used to create fuzzy rules for function approximation purposes, the selection of consequents must be adapted. We consider now a training set that contains patterns $(\mathbf{p}, t)$ with $\mathbf{p} \in \mathbb{R}^n$ and $t \in \mathbb{R}$, i.e. we want to approximate an unknown function $f : \mathbb{R}^n \to \mathbb{R}$, $f(x_1, \ldots, x_n) = y$ based on the training data. We have n input variables $x_i \in X_i \subseteq \mathbb{R}$ and one output variable $y \in [y_{\min}, y_{\max}] \subset \mathbb{R}$ with the range $y_r = y_{\max} - y_{\min}$. We consider one output dimension only for the sake of simplicity. The algorithm can be easily extended to create fuzzy rules with multiple output values.

In the following we present the NEFPROX learning procedure (Algorithm 4.4) [Nauck and Kruse, 1998c]. To determine consequent fuzzy sets, it computes a weighted average $\bar{t}$ (see Eq. 4.2) of the target output values for all patterns that have non-zero membership with an antecedent A discovered in the data. The consequent fuzzy set C that completes a rule $R = (A, C)$ is either picked from exisiting fuzzy sets for the output variable y, or it is generated such that $C(\bar{t}) = 1$. The kind of membership function (e.g. triangular, trapezoidal, bell-shaped) that is generated depends on the preference of the user. A new output fuzzy set will be generated, if there is no available output fuzzy set ν with $\nu(\bar{t}) > \theta$, where θ is a user-defined threshold value. Usually $\theta = 0.5$ is selected and all newly generated fuzzy sets are of the same kind and have a fixed width for their supports. Thus a fuzzy partition of equidistant fuzzy sets with $\forall y : \sum_\nu \nu(y) = 1$ is created automatically. It is also possible that the fuzzy partition of the output variables is given in advance, as it is for the input variables.

In order to be able to select a subset of the created rules for the final rule base, we need a performance measure for the rules. We consider a rule to be useful, if the patterns it covers all have very similar output values, and if the rule covers many patterns. We use a weighted average and variance to determine the performance of

Algorithm 4.2: Select the best rules for the rule base

SelectBestRules

(The algorithm determines a rule base by selecting the best rules from *)*
(the list of rule candidates created by Algorithms 4.1 or 4.4. *)*

```
1:  k = 0; stop = false;
2:  repeat
3:     R' = argmax_R {P_R};
4:     if fixed rule base size then
5:        if (k < k_max) then
6:           add R' to rule base;
7:           delete R' from list of rule candidates;
8:           k = k + 1;
9:        else
10:          stop = true;
11:       end if
12:    else if (all patterns must be covered) then
13:       if (R' covers some still uncovered patterns) then
14:          add R' to rule base;
15:          delete R' from list of rule candidates;
16:          if (all patterns are now covered) then
17:             stop = true;
18:          end if
19:       end if
20:    end if
21: until stop
```

Algorithm 4.3: Select the best rules per class for the rule base

SelectBestRulesPerClass

(The algorithm determines a rule base by selecting the best rules for each *)*
(class from the list of rule base candidates created by Algorithm 4.1. *)*

```
 1: k = 0; stop = false;
 2: repeat
 3:    for all classes C do
 4:       if (∃R : con(R) = C) then
 5:          R' = argmax_{R: con(R)=C} {P_R};
 6:          if (fixed rule base size) then
 7:             if (k < k_max) then
 8:                add R' to rule base;
 9:                delete R' from list of rule candidates;
10:                k = k + 1;
11:             else
12:                stop = true;
13:             end if
14:          else if (all patterns must be covered) then
15:             if (R' covers some still uncovered patterns) then
16:                add R' to rule base;
17:                delete R' from list of rule candidates;
18:             end if
19:             if (all patterns are now covered) then
20:                stop = true;
21:             end if
22:          end if
23:       end if
24:    end for
25: until stop
```

a rule. Note that for the following computations we assume $\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p}) > 0$.

$$\bar{t} = \frac{\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})\cdot t}{\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})}, \tag{4.2}$$

$$\begin{aligned}\mathrm{var}(R) &= \frac{\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})\cdot (t-\bar{t})^2}{\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})} \qquad (4.3)\\ &= \frac{1}{\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})}\left(\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})\cdot t^2 - \frac{1}{\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})}\cdot\left(\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})\cdot t\right)^2\right),\end{aligned}$$

$$P_R = \frac{y_r \cdot \sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})}{\left|\tilde{\mathcal{L}}\right|\cdot\left(2\sqrt{\mathrm{var}(R)}+y_r\right)}, \text{ with } P_R\in[0,1]. \tag{4.4}$$

The performance value P_R for some rule R approaches 0, if $\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})$ approaches 0. For $\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p}) = s$ (the rule R is completely valid for all patterns), the performance measure becomes 1, if $\mathrm{var}(R) = 0$.

The NEFPROX rule learning algorithm uses the following notations:

- $\tilde{\mathcal{L}}$: a set of training data (fixed learning problem) with $\left|\tilde{\mathcal{L}}\right| = s$, which represents a function approximation problem, where patterns $\mathbf{p}\in\mathbb{R}^n$ must be mapped to target values $t\in\mathbb{R}$.

- $R = (A, C)$: a fuzzy rule with antecedent $\mathrm{ant}(R) = A$ and consequent $\mathrm{con}(R) = C$, where $A = (\mu_{j_1}^{(1)},\ldots,\mu_{j_n}^{(n)})$ and $C : Y \to [0,1]$ is a fuzzy set. We use both $R(\mathbf{p})$ and $A(\mathbf{p})$ to denote the degree of fulfilment of rule R (with antecedent A) for pattern $\mathbf{p}$, i.e. $R(\mathbf{p}) = A(\mathbf{p}) = \min\{\mu_{j_1}^{(1)}(p_1),\ldots,\mu_{j_n}^{(n)}(p_n)\}$.

- $\mu_j^{(i)}$: jth fuzzy set of the fuzzy partition of input variable x_i. There are q_i fuzzy sets for variable x_i.

- w_A: variable to compute $\sum_{(\mathbf{p},t)\in\tilde{\mathcal{L}}} R(\mathbf{p})$ for rule R with antecedent A.

- m_A: variable to compute the weighted average $\bar{t}$ (4.2) for rule R with antecedent A.
- v_A: variable to compute var(R) (4.3) for rule R with antecedent A.
- P_R: the performance (4.4) of rule R.

The NEFPROX learning algorithm can be easily extended to fuzzy rules with multiple output variables. In this case the performance measure (including mean and variance) must be computed for each output variable individually. The performance of a rule is than determined by the mean of the performance values over all output variables.

The structure-oriented fuzzy rule learning algorithms presented in this section are very fast, because they only need to processes the training data twice to determine all candidates for a rule base. The selection of the rules to be included into the rule base is guided by a performance measure. The number of rules can be determined automatically such that for each training pattern there is a least one rule with non-zero degree of fulfilment or the number of rules is restricted by some value given by the user. The latter method does not need to process the training data again. Only if the the rule base size is determined automatically, must the training patterns be processed again until so many rules have been selected that all patterns are covered by rules.

If the number of rules is restricted the rule learning algorithm is not very much influenced by outliers. Rules that are only created to cover outliers have a low performance value and will not be selected for the rule base.

The performance of the selected rule base depends on the fuzzy partitions that are provided for the input (and output) variables. To increase the performance the fuzzy sets should be tuned by one of the algorithms discussed in Sections 5.3 and 5.4.

4.3 Handling Symbolic Data

In this section we consider data analysis problems where the data contains both numeric and symbolic information. Instead of representing symbols numerically we want to use symbolic information directly. Fuzzy systems can easily do that, because internally they only process degrees of membership.

We call fuzzy rules that contain symbolic and numeric variables *mixed fuzzy rules*, in order to distinguish them from fuzzy rules that use only numeric variables as is usually the case in the application of fuzzy systems. We discuss an algorithm that can create a rule base of mixed fuzzy rules. The algorithm is an extension to the procedure presented in Section 4.2.

Algorithm 4.4: The NEFPROX rule learning algorithm

1: **for all** patterns $(\mathbf{p}, t) \in \tilde{\mathcal{L}}$ **do** *(* there are s training patterns *)*
2: **for all** input features x_i **do**
3: $\mu_{j_i}^{(i)} = \underset{\mu_j^{(i)}, j \in \{1,\ldots,q_i\}}{\operatorname{argmax}} \{\mu_j^{(i)}(p_i)\}$;
4: **end for**
5: Create antecedent $A = (\mu_{j_1}^{(1)}, \ldots, \mu_{j_n}^{(n)})$;
6: **if** ($A \notin$ list of antecedents) **then**
7: add antecedent A to list of antecedents;
8: $m_A = 0$; $v_A = 0$; $w_A = 0$;
9: **end if**
10: **end for**
11: **for all** patterns $(\mathbf{p}, t)$ of $\tilde{\mathcal{L}}$ **do** *(* compute weighted means and variances *)*
12: **for all** $A \in$ list of antecedents **do** *(* for all antecedents *)*
13:
14: **if** $(A(\mathbf{p}) > 0)$ **then**
15: $w_A = w_A + A(\mathbf{p})$;
16: $m_A = m_A + A(\mathbf{p}) \cdot t$;
17: $v_A = v_A + A(\mathbf{p}) \cdot t^2$;
18: **end if**
19: **end for**
20: **end for**
21: **for all** $A \in$ list of antecedents **do** *(* select consequents for all antecedents *)*
22: $v_A = \frac{1}{w_A} \cdot (v_A - \frac{m_A^2}{w_A})$; $m_A = \frac{m_A}{w_A}$; *(* complete computation of statistics *)*
23:
24: **if** $(\exists \nu : \nu(m_A) > \theta)$ **then**
25: $C = \underset{\nu}{\operatorname{argmax}} \{\nu(m_A)\}$;
26: **else**
27: generate new output fuzzy set C with $C(m_A) = 1$;
28: **end if**
29: create rule R with antecedent A and consequent C;
30: add R to list of rule base candidates;
31: $P_R = \frac{w_A \cdot y_r}{s \cdot (2\sqrt{v_A} + y_r)}$; *(* performance of rule R *)*
32: **end for**
33: SelectBestRules; *(* see Algorithm 4.2 *)*

Mixed fuzzy rules demand the use of fuzzy sets for symbolic variables that cannot be represented by the usual parameterized membership functions like triangles or trapezoids.

We consider two attributes x and y, where $x \in X \subseteq \mathbb{R}$ is numeric and $y \in Y = \{A, B, C\}$ is symbolic. In a fuzzy rule we describe values of x by linguistic terms. We use *lvalue* to denote any such linguistic term (*lvalue* may be a term like *small, approximately zero, large*, etc.). In a mixed fuzzy rule using two variables we can find, for example, the following situations:

(i) fuzzy-exact: if x is *lvalue* and y $= A$ then ...

(ii) fuzzy-imprecise: if x is *lvalue* and y $\in \{B, C\}$ then ...

(iii) fuzzy-fuzzy: if x is *lvalue* and y is $\{(A, \mu(A)), (B, \mu(B)), (C, \mu(C))\}$ then ...

In the first two cases the symbolic variable y has a "switching function" for a rule. If y does not assume one of the values noted in the respective y-term of the antecedent, the rule is not applicable at all. But if y does assume any of these values, the applicability of the rule is not restricted by this argument, and the degree of fulfilment only depends on the value for x.

In the third situation, we use a fuzzy set to describe the value that y may assume, by simply attaching a degree of membership to each element of Y using some membership function $\mu : Y \to [0, 1]$. By giving some value to y we can now restrict the applicability of the rule to any degree between 0 and 1.

Obviously case (i) and (ii) are just special cases of case (iii), because we can replace $y = A$ by y is $\{(A, 1), (B, 0), (C, 0)\}$ and $y \in \{A, B\}$ by y is $\{(A, 1), (B, 1), (C, 0)\}$.

Because the elements of Y are not ordered, we cannot easily use a linguistic term to label fuzzy sets like $\{(A, \mu(A)), (B, \mu(B)), (C, \mu(C))\}$. This means the interpretability in terms of the variables is restricted compared to fuzzy rules that just use variables on metric scales. We will discuss this issue at the end of this section.

For cases (i) and (ii) we can remove the symbolic variable from the fuzzy rules, and create different rule bases, one for each combination of values of y (see Figure 4.7). Depending on the value of y we simply select the applicable rule base. Such a situation may be, for example, useful in a medical setting, where we want to classify diseases of patients according to certain symptoms. If we assume that the classification depends on whether the patient is female or male, we can simply build two different rule bases – one to classify female patients, and one to classify male patients.

For case (iii) the following example describes the idea of the learning algorithm.

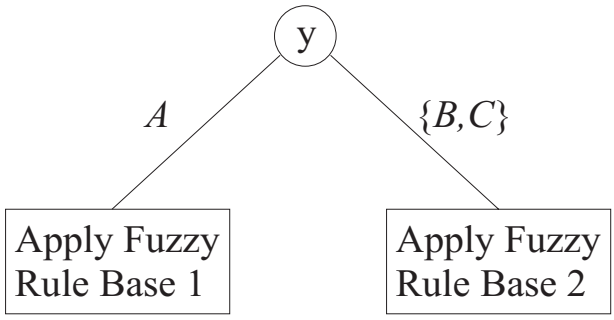


Figure 4.7: Switching between fuzzy rule bases depending on a symbolic variable

Example 4.1 We consider an artifical data set given in Figure 4.8. We partition x by three fuzzy sets labelled *small, medium* and *large*. The fuzzy sets are represented by triangular membership functions.

The degrees of membership for any value adds up to 1.0, i.e. the membership functions overlap at degree 0.5. The vertical lines in Figure 4.8 visualize these points. The values of y are represented on the vertical scale, but note that the scale is nominal, i.e. the ordering of elements has no meaning.

We assume that the data can be classified into a *positive* class and a *negative* class, as denoted by the + and − signs in the data space. As can be seen in Figure 4.8 there is some degree of overlapping between both classes, especially in the area where x is *small*.

To create a rule base from this data we first consider the situation, where x is *small* and the class is *positive*. There are 3 cases where $y = A$, 2 cases where $y = B$ and 5 cases where $y = C$. By normalizing these values, we obtain the fuzzy set $\{(A, 0.6), (B, 0.4), (C, 1.0)\}$. If we consider the *negative* class and x is *small*, we obtain the fuzzy set $\{(A, 0.33), (B, 1.0), (C, 0.33)\}$ in the same manner.

This means, we can add the following two rules R_1 and R_2 to the rule base:

R_1 : if x is *small* and y is $\nu_1 = \{(A, 0.6), (B, 0.4), (C, 1.0)\}$,
then the class is *positive*

R_2 : if x is *small* and y is $\nu_2 = \{(A, 0.33), (B, 1.0), (C, 0.33)\}$,
then the class is *negative*

◇

To study the application of the two rules created in Example 4.1, we consider three patterns where $x = x_0$ in each case with $\mu_{\text{small}}(x_0) = 1.0$ and y having a different value in each case (let $\mu_{\text{small}} : \mathbb{R} \to [0, 1]$ be the membership function to represent *small*). Table 4.1 gives the degrees of fulfilment for both rules and the classification for all three cases.

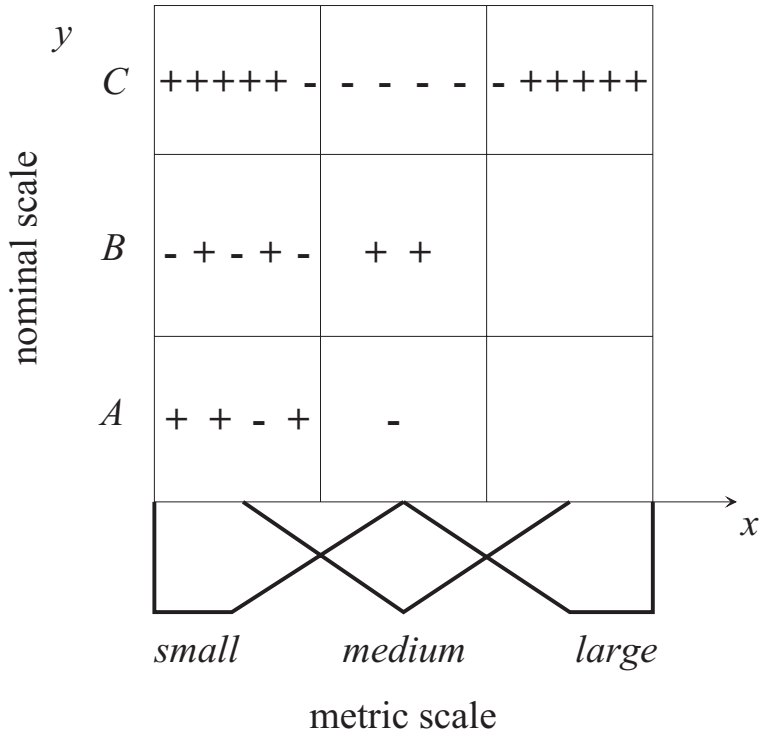


Figure 4.8: An artificial data set with a numeric and a symbolic attribute

Table 4.1: Classification of three sample cases using rules R_1 and R_2

x	$\mu_{\text{small}}(x)$	y	$\nu_1(y)$	$\nu_2(y)$	R_1	R_2	class
x_0	1.0	A	0.6	0.33	0.6	0.33	*pos.*
x_0	1.0	B	0.4	1.00	0.4	1.00	*neg.*
x_0	1.0	C	1.0	0.33	1.0	0.33	*pos.*

The rules are, like rules that use only metric variables, representations of typical representatives for the classes. If we consider the positive class, a typical case would, for example, have a *small* value for x and either C, A or B for y, where we would consider C more typical than A and A more typical than B. For the *negative* class we would consider B more typical than A or C.

R_1 and R_2 are partially contradictory, as they overlap in all variables. If we use

$$\theta(\mu_1, \mu_2) = \sup_x \min\{\mu_1(x), \mu_2(x)\} \tag{4.5}$$

to denote the degree of similarity or overlapping of two fuzzy sets μ_1 and μ_2, then we obtain $\theta = 0.4$ for the antecedents of rules R_1 and R_2. Rules with $\theta = 0$ are mutually exclusive, and for $\theta = 1$ the rules are either identical or one rule is a generalization of

the other (if the consequents are identical), or they are completely contradictory (if the consequents are different). Partial contradiction is, however, common for fuzzy rule bases, as overlapping of rules is a desired feature.

Example 4.1 (contd.) If we use the rule generation technique further, we now obtain the following two rules:

R_3 : if x is *medium* and y is $\nu_3 = \{(A, 0.25), (B, 0.0), (C, 1.0)\}$, then the class is *negative*,

R_4 : if x is *medium* and y is $\nu_4 = \{(A, 0.0), (B, 1.0), (C, 0.0)\}$, then the class is *positive*.

For R_3 and R_4 we obtain $\theta = 0$, i.e. they are mutually exclusive. In this case it would be possible to use crisp sets to describe the values for y ($y \in \{A, C\}$ for R_3 and $y = B$ for R_4). However, we would lose the information that the combination x is *medium* and $y = C$ is much more typical for the *negative* class than the combination x is *medium* and $y = A$. It is therefore useful to keep the fuzzy set representation.

For the last box in Figure 4.8 that contains data, we obtain two contradictory rules ($\theta = 1$):

R_5 : if x is *large* and y is $\nu_5 = \{(A, 0.0), (B, 0.0), (C, 1.0)\}$, then the class is *positive*,

R_6 : if x is *large* and y is $\nu_6 = \{(A, 0.0), (B, 0.0), (C, 1.0)\}$, then the class is *negative*.

We cannot include both rules in the rule base, as we can only tolerate partial contradiction. Therefore we keep the rule with better performance, i.e. we delete the rule that would cause more misclassifications. In this case we delete rule R_6 and keep rule R_5. The final rule base consists therefore of rules $R_1, \ldots, R_5$. ◇

The rule learning procedure is given in Algorithm 4.5. It computes a fuzzy rule base from a set of training data containing symbolic and numeric data.

Algorithm 4.5 uses the following notations:

- $\tilde{\mathcal{L}}$: a set of training data (fixed learning problem) with $\left|\tilde{\mathcal{L}}\right| = s$, which represents a classification problem where patterns $\mathbf{p}$ are to be assigned to m classes $C_1, \ldots, C_m$.
- $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$: a training pattern consists of an input vector $\mathbf{p} \in X_1 \times \ldots \times X_n$ and a target vector $\mathbf{t} \in [0, 1]^m$. $\mathbf{p}$ consists of u numeric and v symbolic features ($u + v = n$), i.e. X_i is either a subset of $\mathbb{R}$ or a (finite) set of symbols. The target vector represents a possibly vague classification of the input pattern $\mathbf{p}$. The class index of $\mathbf{p}$ is given by the index of the largest component of $\mathbf{t}$: $\text{class}(\mathbf{p}) = \text{argmax}_j\{t_j\}$.

- $R = ((A, M), C)$: a fuzzy classification rule with antecedent $\text{ant}(R) = (A, M)$ and consequent $\text{con}(R) = C$, which denotes a class. $A = (\mu_{j_1}^{(1)}, \ldots, \mu_{j_u}^{(u)})$ is created by fuzzy sets for numeric variables and is the first part of the antecedent. $M = (\mathbf{m}_{j_1}^{(1)}, \ldots, \mathbf{m}_{j_v}^{(v)})$ is the second part of the antecedent and is created by the fuzzy sets for the symbolic variables.
- $\mu_j^{(i)}$: jth fuzzy set of the fuzzy partition of input variable x_i. There are q_i fuzzy sets for variable x_i.
- $\mathbf{m}_j^{(k)}$: jth fuzzy set of the kth symbolic variable x_k. There are m fuzzy sets for each symbolic variable, i.e. one fuzzy set per class. A fuzzy set $\mathbf{m}_j^{(k)}$ is represented by a vector that contains the degrees of membership for all elements of X_k. At the time of initialization (Algorithm 4.5, line 12) all entries of $\mathbf{m}_j^{(k)}$ are set to zero. We use $\mathbf{m}_j^{(k)}[x]$ to denote that the degree of membership for x is accessed for manipulation (Algorithm 4.5, line 22)

The algorithm starts by creating initial antecedents that contain only numeric attributes using the procedure described in Section 4.2 (Algorithm 4.1). After the training data is processed once, all k antecedents that are supported by the data have been found. In the next step, from each antecedent m rules are created, one for each class, and the initial antecedents are completed by constructing fuzzy sets for the symbolic attributes as shown in Example 4.1. This means there is now an initial rule base that contains a set of $m \cdot k$ rules. This rule set can be inconsistent, because it can contain contradictory rules. After resolving inconsistencies, by selecting the rule with a better performance from multiple rules with identical antecedents but different consequents, a final list of rule base candidates is created. Then one of the rule evaluation algorithms given in Section 4.2 is applied to select a final rule base [Nauck et al., 1999].

After rule creation the fuzzy sets of both numeric and symbolic variables can be trained to improve the performance of the classifier by using the algorithms given in Section 5.4.

The mixed fuzzy rules created by Algorithm 4.5 cannot be as easily interpreted as fuzzy rules that use only numeric variables and continuous membership functions, which can be labelled with terms like *small* or *large*.

Fuzzy sets that are denoted as an ordered list of pairs are hard to be labelled linguistically. In some cases linguistic labels can be found by inspection. For example, if we have a symbolic variable describing the job of a person the fuzzy set {(accountant, 0), (consultant, 0.3), (engineer, 0.7), (lecturer,1), (professor, 1)} may be labelled by *academic job*.

If fuzzy rules are created by learning, then it is useful to also create linguistic labels automatically. To quickly generate a rough linguistic term for a fuzzy set given by an ordered list of pairs we could use "y is A or C or B" for y is $\{(A, 1.0), (B, 0.4), (C, 0.7)\}$.

Algorithm 4.5: Learning mixed fuzzy rules from numeric and symbolic data

1: **for all** $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$ **do** *(* find all hyperboxes that contain data *)*
2: **for all** metric input features x_i **do**
3: $\mu_{j_i}^{(i)} = \underset{\mu_j^{(i)}, j \in \{1,\ldots,q_i\}}{\operatorname{argmax}} \{\mu_j^{(i)}(p_i)\}$;
4: **end for**
5: Create $A = (\mu_{j_1}^{(1)}, \ldots, \mu_{j_n}^{(n)})$; *(* First part of the antecedent *)*
6:
7: **if** ($A \notin$ list of antecedents) **then**
8: add A to list of antecedents;
9: **end if**
10: **end for**

11: **for all** $A \in$ list of antecedents **do** *(* create rule base candidates *)*
12: initialize M;
13: create complete antecedent (A, M);
14: **for all** classes C **do**
15: create rule $R = ((A, M), C)$ and add it to list of rule base candidates;
16: **end for**
17: **end for**

18: **for all** $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$ **do** *(* compute frequencies of symbolic variables *)*
19: **for all** $R \in$ list of rule base candidates **do**
20: **if** (class($\mathbf{p}$) = con(R)) **then**
21: **for all** symbolic features x_k **do**
22: with R do: $\mathbf{m}^{(k)}[p_k] = \mathbf{m}^{(k)}[p_k] + 1$;
23: **end for**
24: **end if**
25: **end for**
26: **end for**

27: **for all** $R \in$ list of rule base candidates **do**
28: with R do: normalize all $\mathbf{m}^{(i)}$; *(* transform the $\mathbf{m}^{(i)}$ into fuzzy sets *)*
29: compute the performance P_R of R; *(* see Eq. 4.1 and Algorithm 4.1 *)*
30: **end for**

31: Find all contradicting rules and resolve conflicts;
32: **if** (select best rules) **then**
33: SelectBestRules; *(* see Algorithm 4.2 *)*
34: **else if** (select best rules per class) **then**
35: SelectBestRulesPerClass; *(* see Algorithm 4.3 *)*
36: **end if**

The order in which the feature values with non-zero membership are listed, expresses the preferences represented in the degrees of membership. In this case we learn from the label, that A is more typical than C and C is more typical than B. If we need to know the exact degrees of membership, we can look at the fuzzy set.

This interpretation is similar to common linguistic labels like *approximately zero* for a numeric variable. In this case we also know, that 0 is the most typical value for the variable and larger or smaller values are less typical. If we are interested in the exact degrees, we also have to look at the membership function.

4.4 Treatment of Missing Values

Missing values are common in many practical settings. It is not always possible to observe all features of a pattern. This can be due to high costs, faulty sensors, errors in recording, etc. If a feature is sometimes measured and sometimes not, we can use the cases for which it has been measured to learn to predict its values when it is not. In decision tree learning, for example, the probability distribution of the feature is used when a value is missing [Quinlan, 1993]. Another approach to learning in the presence of unobserved variables is the EM algorithm (estimation and maximization) [Dempster et al., 1977, Mitchell, 1997]. The EM algorithm searches for a maximum likelihood hypothesis by repeatedly re-estimating the expected values of the unobserved variables given the current hypothesis, then recalculating the maximum likelihood hypothesis using these expected values [Mitchell, 1997].

Other approaches to deal with missing data [Hair et al., 1998] are

(i) to use only cases with complete data

(ii) to delete cases and/or variables with missing data with excessive levels

(iii) to use imputation methods that replace missing values with a constant, the mean, a value computed by regression, etc.

The first option is usually not feasible, as it may turn out that too many cases cannot be used for training and the training set becomes too small. In real world data it is easily possible that each case and each variable displays at least one missing value.

The second option can be used to clean up the training data, i.e. to remove case and/or variables, if they have a certain percentage of missing values that makes them unusable for further analysis. However, this does not solve the problem completely as there are cases left that still have (some) missing values.

Imputation methods require a thorough analysis of the data and why missing values occur. Blind imputation by some constant or a mean value might actually make the situation worse. For example, it may be possible that a value of some variable

cannot be measured due to a faulty sensor, if the variable exceeds some threshold. If in this case simply the mean is used whenever a value is missing, the distribution of the considered variable becomes biased.

If a large part of the data is missing, then the reason for that should be carefully analysed during the pre-processing stage of the data analysis process. However, often the percentage of missing values is low, and expensive missing values analysis is not possible.

We therefore use the following simple strategy for learning fuzzy rules. If a feature is missing, we do not make any assumptions about its real value but rather assume that any value may be possible. Based on this assumption we do not want to restrict the application of a fuzzy rule to a pattern with missing features. This means a missing value will not influence the computation of the degree of fulfilment of a rule. This can be done by assigning 1.0 as the degree of membership to the missing feature [Berthold and Huber, 1997], i.e. a missing value has a degree of membership of 1.0 with any fuzzy set. A pattern where all features are missing would then fulfil any rule of the fuzzy rule base with a degree of 1.0, i.e. any class would be possible for such a pattern. We denote a pattern with missing values by $\mathbf{p} = (\mathbf{x}, ?)$. We compute the degree of fulfilment τ_r of some rule R_r by

$$\tau_r(\mathbf{x}, ?) = \min_{x_i} \{\mu_r^{(i)}(x_i), 1\} = \min_{x_i} \{\mu_r^{(i)}(x_i)\}. \tag{4.6}$$

In learning a fuzzy system from data there are three stages where missing values must be considered:

(i) learning fuzzy rules,

(ii) training membership functions,

(iii) application of the fuzzy system.

Item (iii) has been considered above. [Berthold and Huber, 1997] suggested completing an input pattern with missing values by using the fuzzy rule base of the fuzzy system during training. We will not use this approach here, because it cannot be used for rule learning and we want to use the same technique in all three stages. In the following we consider the treatment of missing values in structure-oriented rule learning. Hyperbox-oriented and cluster-oriented approaches are mentioned at the end of this section.

Rule learning, as described in Section 4.2, consists of three steps:

(i) determine all possible antecedents,

(ii) create an initial rule base by finding an appropriate consequent for each antecedent,

(iii) select a final rule base from the initial rule base by computing the performance of each rule.

Step (i) is implemented by selecting hyperboxes from a structured data space. If we encounter a missing value, any fuzzy set can be included in the antecedent for the corresponding variable. Thus we create all combinations of fuzzy sets that are possible for the current training pattern.

Example 4.2 Consider the situation in Figure 4.9. We use three fuzzy sets to partition each variable. If we obtain the input pattern $(x_0, ?)$ as shown in Figure 4.9 we can assign the fuzzy set *large* to the first feature, because it yields the largest degree of membership for x_0. As the value for y is missing, any fuzzy set is possible for y. Therefore we create the antecedents (x is *large* and y is *small*), (x is *large* and y is *medium*), and (x is *large* and y is *large*). In step (ii) of the rule learning algorithm appropriate consequents will be determined for these antecedents, depending on all training patterns. In step (iii) the rules with the highest performance will be selected. ◇

Algorithm 4.6 provides a loop to create antecedents, if the training data contains missing values. This algorithms can be used as a template to modify the loops in Algorithm 4.1 (lines 1–9) and Algorithm 4.4 (lines 1–10) in order to enable those algorithms to handle missing values. The notation used in Algorithm 4.6 is the same as in the other two algorithms (Section 4.2).

After a rule base has been created, the membership functions are trained by one of the algorithms discussed in Chapter 5. If a missing value is encountered in a variable, then for the corresponding fuzzy set simply no training signal will be generated by this pattern.

To enable hyperbox-oriented approaches to handle missing values it must be possible to create a hyperbox that may have completely undefined parameters for a dimension where a missing value occured. Undefined parameters will obtain a value when a pattern with no missing value in the corresponding dimension is encountered. In addition, the computation of the degree of membership for hyperboxes must be changed according to (4.6) and a hyperbox is not changed (reduced or enlarged) in a dimension, if the corresponding value is missing.

When fuzzy cluster analysis is used to create fuzzy rules the treatment of missing values is more difficult, as the degree of membership for a pattern depends on the distance to a cluster prototype. Therefore several imputation methods are discussed for fuzzy clustering [Timm and Klawonn, 1998, Timm and Kruse, 1998].

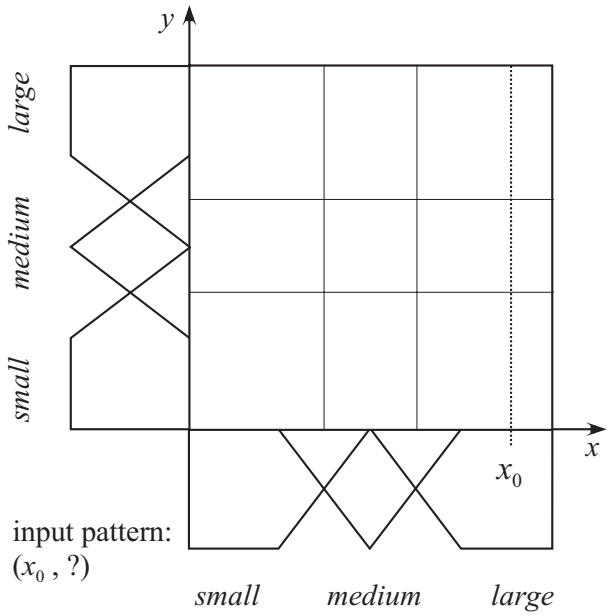


Figure 4.9: Rule learning with missing values: three rules are created by the pattern $(x_0, ?)$, because for y three fuzzy sets are possible

---

Algorithm 4.6: Creating antecedents when there are missing values

---

for each pattern $(\mathbf{p}, \mathbf{t})$ of $\tilde{\mathcal{L}}$ **do** *(* there are s training patterns *)*
  create a new empty antecedent A';
  for all input features x_i whose value is not missing **do**
    $\mu_{j_i}^{(i)} = \underset{\mu_j^{(i)}, j \in \{1,\ldots,q_i\}}{\operatorname{argmax}} \{\mu_j^{(i)}(p_i)\}$;
    add $\mu_{j_i}^{(i)}$ to antecedent A';
  end for
  repeat
    create a new empty antecedent A;
    $A = A'$;
    create a new combination of fuzzy sets from all
    missing input features and add them to A;
    if ($A \notin$ list of antecedents) **then**
      add antecedent A to list of antecedents;
    end if
  until all combinations were enumerated
end for

4.5 Analysis of the Learning Algorithms

In this section we analyze the time and memory complexity of the learning algorithms that are discussed in this chapter. We make the following assumptions.

(i) We use a training data set $\tilde{\mathcal{L}}$ with $n > 1$ input variables, $m > 1$ possible output values (output fuzzy sets or class labels) and $\left|\tilde{\mathcal{L}}\right| = s > 1$.

(ii) The final rule base has $r \geq 1$ rules.

(iii) We select

$$q = \max\{q_1, \ldots, q_n,\ m\} > 1 \tag{4.7}$$

to represent the number of fuzzy sets for each input variable x_i and the number of output values, respectively. Because we are interested in linguistically interpretable fuzzy systems, we will only expect small values for q, for example, $q \leq 10$.

(iv) For the number of antecedents Q that can be discovered in the data we have

$$Q = \min\{\prod_{i=1}^{n} q_i,\ s\} \leq \min\{q^n,\ s\},$$

i.e. Q increases exponentially with the number of variables, but it cannot exceed the number of patterns. The actual number of antecedents that are created by the training data is given by $k \leq Q$.

When we analyze the runtime behavior of the rule learning algorithms we consider the following operations:

- computation of degrees of membership,
- computation of degrees of fulfilment,
- comparisions of antecedents or performance values.

We begin our considerations with the rule learning algorithms 4.1 and 4.4, which need two cycles through the training data. To identfy the antecedent for each pattern snq computations of a degree of membership are required.

To find out, whether an antecedent is already included in the list of antecedents, $Q(Q-1)/2$ comparisions are required in the worst case, if we assume that each pattern creates a new antecedent. We can stop the creation of antecedents, if the list actually contains Q entries. However, in many cases data will have a clustered

Table 4.2: The number of antecedents found in the Iris data (150 patterns, 4 variables) depending on the number of fuzzy sets per variable

q	2	3	4	5	6	7	8
k	8	20	35	47	61	67	89
q^4	16	81	256	625	1296	2401	4096

(q: number of fuzzy sets, k: number of antecedents)

structure and cover only parts of the domain. In these cases the number of antecedents that are found in the data is much less than the theoretically possible number.

Because each antecedent can be represented by the indices of its fuzzy sets, a hash table can be conveniently used to store the antecedents. We can assume that for each pattern one application of the hash function is required in order to determine if the antecedent that was created by the current pattern is already stored (we ignore collisions). This means that altogether we need Q calls of the hash function. Because we are interested in an upper bound of the compuational complexity, we assume $Q = s$. This is also reasonable, if we consider high-dimensional real world data sets where we expect $s < q^n$ in most of the cases.

An example for the number of antecedents to be found in a data set depending on the number of fuzzy sets per variables is given in Table 4.2. We used the Iris data set [Fisher, 1936], which contains 150 patterns with 4 variables. NEFCLASS-J (see Chapter 6) was used to create antecedents with triangular fuzzy sets. We can see that the number of antecedents k is always much smaller than the number of patterns, even if the number of different antecedents given by q^4 becomes very large, resulting in a fine granularity of the data space.

To determine the best consequent for each antecedent the patterns must be processed again. We need to compute sk degrees of fulfilment, i.e. skn degrees of membership, to collect the necessary statistical information. Afterwards we need another kq iterations to assign a consequent to each antecedent and to compute the performance measure for each created rule.

Selecting a final rule base from the list of rule base candidates requires rk comparisions of performance values, if the number r of rules to be selected is fixed. If enough rules must be selected so that all patterns are covered, then $rk + sn$ operations are necessary, because for each selected rule the patterns with a non-zero degree of fulfilment must be determined. In this case the number of selected rules r depends on the training data. It is only necessary to compute s degrees of fulfilment, because patterns that are already covered can be marked and need not be processed again.

From our considerations above we can see that a rule base is available after $f_t(s, n, q, k, r)$ operations with

$$\begin{aligned} f_t(s, n, q, k, r) &\leq snq + s + skn + kq + rk + sn \\ &\leq sn(q + k + 1) + s + k(q + r). \end{aligned} \tag{4.8}$$

In this expression k and r are both bounded by s. Because want to have more training data than free parameters, it is also reasonable to assume $s \geq nq$ for classification problems and $s \geq nq + q$ for function approximation problems. From these considerations we obtain an upper bound for the number of operations with

$$f_t(s, n, q, k, r) \leq 3s^2 + s(n + q + 1) = O(s^2). \tag{4.9}$$

To obtain a lower bound for the complexity, we consider a classification problem and a function approximation problem. Because we now want to obtain a lower bound for the number of operations, we set

$$q = \min\{q_1, \ldots, q_n,\ m\}. \tag{4.10}$$

For a classifier we need at least $r = q$ rules, because we assume that there are no empty classes. We assume that also only $k = q$ rules were detected in the training data. This requires that we use at least $s = q$ training patterns. However, because the number of free parameters is at least nq we do not accept training data sets with less then nq patterns, i.e. we assume $nq \leq s$. We also do not need to select a final rule base, because it is already minimal. From these considerations and with (4.8) we obtain a lower bound for the number of operations such that

$$f_t(s, n, q, k, r) \geq 2n^2q^2 + nq + q^2 = O(n^2q^2). \tag{4.11}$$

For a function approximation problem, it can be possible that only one rule is detected during learning, i.e. $r = k = 1$. We again assume $nq < s$ and with (4.8) we obtain

$$f_t(s, n, q, k, r) \geq n^2q^2 + n^2q + nq + q = O(n^2q^2). \tag{4.12}$$

We conclude that for the time complexity of the structure-oriented rule learning algorithms 4.1 and 4.4 we have to expect at least $O(n^2q^2)$ if we only use a minimal number of training patterns and the solution contains a minimal number of fuzzy rules. We have to expect a complexity of $O(s^2)$ for data-rich applications where many rules are generated.

If we use a modified version of Algorithm 4.4 to create fuzzy systems for a function approximation problem with $v > 1$ output variables, the term kq in (4.8) must be replaced by vkq. However, we still obtain $O(s^2)$ by assuming $s \geq vq$, because the number of free parameters is in this case $nq + vq$.

If we use training data with missing values and apply Algorithm 4.6, a pattern with missing values can create multiple rules at once. However, the number of rules is still bound by s, i.e. the complexity does not change.

If the training data contains numeric and non-numeric attributes, we apply the variant of Algorithm 4.1 given by Algorithm 4.5. In this case we have $n = n_1 + n_2$ variables, where n_1 is the number of numeric attributes and n_2 the number of non-numeric attributes. We again use q as it is given by (4.7), where q_i is either a number of fuzzy sets or a number of symbolic attribute values. The algorithm needs $sn_1q + s$ operations to find k antecedents by using only the metric variables. Then kq rule base candidates are created in lines 11–17 of Algorithm 4.5. We interpret the initialization of the fuzzy sets for non-numeric variables, the completion of the antecedent and the creation of a rule as one operation only. The part of the algorithm between lines 18 and 26 requires $skqn_2$ operations. This part increases the complexity, because in this part it is possible that more than s rule base candidates must be processed.

The normalization of the fuzzy sets of the non-numeric variables requires kqn_2q operations and the computation of the performance values is done in $skqn$ steps. Conflict resolution (line 31) requires $kq(kq+1)/2$ comparisions of rules and selection of a final rule base does not require more than $rkq + sn$ steps. If we again use the estimations $k \leq s$, $r \leq s$, $nq \leq s$, we obtain

$$\begin{aligned} f_t(s, n_1, n_2, q, k, r) &\leq sn_1q + s + kq + skqn_2 + kqn_2q + skn \\ &\quad + kq(kq+1)/2 + rkq + sn \\ &\leq s^2 + s + sq + s^3 + s^3 + s^2n \\ &\quad + (s^2+s)/2 + s^2 + sn \\ &= O(s^3). \end{aligned} \tag{4.13}$$

In the simplest case we have only non-numeric attributes, i.e. $n = n_2$ and $n_1 = 0$. In this case the algorithm creates only one rule per class, and we can set $k = 1$ in (4.13). We also do not need to check for contradictions and do not need to select a final rule base. If we use the same estimations as for (4.12) and determine q by (4.10), we obtain

$$f_t(s, n, q, k, r) \geq q + n^2q^2 + nq^2 + n^2q = O(n^2q^2). \tag{4.14}$$

We conclude that Algorithm 4.5 is at least as complex as Algorithm 4.1. In the worst case we obtain a time complexity of $O(s^3)$.

The memory requirements of the algorithms are mainly determined by the training data. The data set consists of $s(n + q)$ values. For each variable we must store q fuzzy sets. To describe triangular, trapezoidal or bell-shaped membership functions, 4 parameters are sufficient. If there are symbolic attributes, a fuzzy set is given by q degrees of membership. We assume that q is given by (4.7). Each fuzzy rule requires

n indices of fuzzy sets and a consequent parameter. The algorithm needs to store $k \leq s$ rule base candidates. We again assume $s > nq$. This means the memory complexity $f_m(s, n, q)$ of the algorithms can be estimated by

$$\begin{aligned} f_m(s, n, q) &\leq s(n + q) + nq^2 + k(n + 1) \\ &= s(n + q) + sq + s(n + 1) = O(sn + sq) \end{aligned}$$

Chapter 5

Optimization of Fuzzy Rule Bases

If we have decided to use a specific fuzzy rule base to model a problem, then we can regard the rule base as structural knowledge about the specific problem expressed in a linguistic way. If the performance of the fuzzy model is not adequate, and if we do not want to change the rule base, then we must modify the fuzzy sets which represent the linguistic terms used in the rules.

The most simple approach to train a fuzzy system is to use adaptive rule weights. However, as we will see in Section 5.1, this can prevent a suitable linguistic interpretation of learning outcomes. Fuzzy systems with rule weights have problems to create understandable models, but they can be useful when accurate prediction is more important than a comprehensible model. Based on the analysis of adaptive rule weights we conclude that it is preferable to train the membership functions if a fuzzy system is going to be used in data analysis.

Section 5.2 contains some general thoughts on training fuzzy sets in different kind of fuzzy systems. If the fuzzy model uses only differentiable functions, then a standard backpropagation algorithm using gradient descent can be applied. This only applies to Sugeno-type fuzzy systems, which are also only partly interpretable and are therefore less useful in data analysis. Gradient descent techniques can usually not be used in Mamdani-type fuzzy systems. Therefore heurisitics other than gradient descent must be developed.

In Sections 5.3 and 5.4 we present neuro-fuzzy learning algorithms for function approximation and classification problems, which can be used for creating Mamdani-type fuzzy systems in data analysis. These algorithms also use constraints to ensure the interpretability of the induced fuzzy model.

Section 5.5 describes methods for optimizing the structure of a fuzzy model by pruning variables and rules in order to obtain a more compact and more interpretable model. The chapter is concluded with an analysis of the complexity of the learning algorithms.

5.1 Adaptive Rule Weights

The most simple way to apply a learning algorithm to a fuzzy system is to use rule weights. A rule weight is used to modify the output of a fuzzy rule by either changing the degree of fulfilment or the output fuzzy set. Weighted fuzzy rules are often used in commercial fuzzy software and also in some neuro-fuzzy models [Berenji, 1992, Kosko, 1992, Zimmermann et al., 1996].

Rule weights add a new computational aspect to the evaluation procedure of a fuzzy rule base and this can lead to some confusion regarding the semantics and the linguistic interpretation of a fuzzy system. This is mainly due to the fact that rule weights often come with some ad-hoc interpretation that hides the real influence of rule weights on a fuzzy system.

In this section we show that a weighted fuzzy rule can be replaced by an equivalent fuzzy rule with modified membership functions. If this is done, the effect of a weight to the semantics of a fuzzy rule becomes apparent. It turns out that weights can implicitly cause membership functions to change in such a way that they can hardly be interpreted linguistically.

Thus, rule weights can completely destroy the interpretability of a fuzzy system. Interpretable fuzzy systems should use normal and convex membership functions that adequately cover the domain of a variable. The main point of interpretability is that a user of a fuzzy system should be able to label each fuzzy set with a suitable linguistic term. Another important issue is that each linguistic term should be represented by only one membership function in the fuzzy system. We will show that if there is an interpretable fuzzy system, then the interpretability is lost once rule weights are used. Formal aspects of fuzzy partitions that ensure interpretability are not within the scope of this discussion (compare Section 3.4).

In fuzzy systems for function approximation applying a rule weight to a fuzzy rule means changing the representation of a multidimensional fuzzy set. The influence of a rule weight on the interpretation of such a modified rule becomes visible in its projections which are the one-dimensional membership functions of the individual variables.

Weighted rules are also considered in probabilistic or possibilistic settings. For example, certainty factors are an early heuristic approach to use rule weights for modeling beliefs. However, it turned out that the original certainty factor approach is inconsistent [Heckerman, 1988, Kruse et al., 1991]. For modeling degrees of belief or truth in fuzzy knowledge-based systems refer e.g. to [Dubois et al., 1989, Dubois and Prade, 1988, Kruse et al., 1994a, Kruse et al., 1991].

Implementation of Rule Weights

A weighted fuzzy rule is often written by appending "with w" to it, where w is a real value that is usually different for each rule, e.g.:

$$R_i\text{: if } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z \text{ is } C \text{ with } w_i,$$

where A, B and C are fuzzy sets of the real line, x and y are input variables, z is the output variable and w_i is the real-valued rule weight of rule R_i. The meaning of the "with-operation" that ties a weight to a rule must be defined. A rule weight could be applied to each intermediate result that is computed during evaluation of a fuzzy rule.

The following steps are carried out to compute the conclusion of a fuzzy rule:

(a) Determine the values of all input variables.

(b) Determine the degree of membership for each input value and each fuzzy set of the fuzzy partition of the respective input variable.

(c) Compute the degree of fulfilment of the rule antecedent.

(d) Compute the conclusion. Depending on the type of consequent, the conclusion can be a fuzzy set or a real value.

A lot of neuro-fuzzy approaches use a neural network-like graph to illustrate the data flow and the computations that are carried out in a fuzzy system (Section 3.3). This neural network representation is then used to formalize the application of a learning algorithm. Many neuro-fuzzy approaches use a 5-layer node-oriented architecture as shown in Figure 3.2(a) on page 32.

Such a 5-layer representation of a fuzzy system is suitable for defining a neural network-like learning algorithm based on adaptive weights that are attached to some of the connections. If learning is to be based on rule weights, then this means that not all connections can be weighted, because there must be only one weight for each rule. The values that are propagated on the connections are the intermediate results presented in the list given above. To study the influence of weights applied to the 5-layer representation of a fuzzy system, we examine the connections between each two adjacent layers.

(a) Input layer → antecedent layer:
On these connections the input values are propagated. Attaching weights to the connections would result in an individual scaling of each input value for each fuzzy rule. This means weights at these connections could not be

interpreted as rule weights as there would be as many weights per rule as there are input values instead of just one weight.

In addition weights at these connections are not suitable as they would scale the domains of the input variables. The fuzzy sets that are stored in the nodes of the antecedent layer are not changed and this means they no longer fit the domain of the variables.

(b) Antecedent layer → rule layer:
On these connections the degrees of membership of the input values are propagated. Weights at these connections would individually scale the degrees of membership before they are combined to the degrees of fulfilment of the rule in the nodes of the rule layer. Weights at these connections could not be interpreted as rule weights for the same reason as in case (a).

Scaling of degrees of membership can also cause semantical problems, for example, if the weights are negative or larger than 1. This issue is discussed below.

(c) Rule layer → consequent layer:
On these connections the degrees of fulfilment of the rules are propagated. Here a rule weight can be attached, as there is exactly one connection per rule. As in case (b) scaling of degrees of fulfilment can cause semantical problems (see discussion below).

(d) Consequent layer → output layer:
On these connections the output fuzzy sets (conclusions) of the fuzzy rules are propagated. Weights at these connections would result in modifying the support of the output fuzzy sets. This is also a possible place to encode rule weights because there is exactly one connection per rule.

Modifying the support of output fuzzy sets results in changing their position in the domain of the output variable. The resulting problems are discussed below.

In general we can obtain an adaptive system, if we attach weights to all connections of the network representation of a fuzzy system and define a suitable learning algorithm. However, from the considerations above we see that rule weights can only be applied to the degree of fulfilment of a rule or to the conclusion of a rule. This means that rule weights can only be represented by weights between the rule layer and the consequent layer (c) or the consequent layer and the output layer (d). In the following we will therefore study only these two cases:

(i) **Rule weights are applied to the degree of fulfilment**: The antecedent of a rule is evaluated to determine a degree of fulfilment which is then multiplied

by a rule weight. For a weighted fuzzy rule R_k from the rule base of a fuzzy system $F_{\mathcal{R}}$ (Definition 3.2) the degree of fulfilment is computed by

$$\tau_k = w_k \cdot \top_1\{\mu_k^{(1)}, \ldots, \mu_k^{(n)}\} \tag{5.1}$$

where $w_k \in \mathbb{R}$ is the weight of R_k.

(ii) **Rule weights are applied to the conclusion**: The degree of fulfilment of a rule is used to compute the conclusion which is then multiplied by a rule weight. If the conclusion is a fuzzy set, its support is changed in this way. Let ν_k be an output fuzzy set of a weighted fuzzy rule R_k. The weighted output fuzzy ν'_k set is computed by

$$\nu'_k = w_k \cdot \nu_k \text{ with } \nu'_k(y) = \sup\{\nu_k(t) | y = w_k \cdot t\} \tag{5.2}$$

where $w_k \in \mathbb{R}$ is the weight of R_k.

Interpretation of Weighted Fuzzy Rules

In the following we will discuss the impact of rule weights on a fuzzy system. We are not interested in how the rule weights are determined, i.e. what the actual learning algorithm looks like. This is not important for the influences of the rule weights.

A rule weight is sometimes interpreted as a measure of "importance", "influence" or "reliability". All these interpretations are usually of an ad-hoc nature and do not actually reflect the impact of rule weights.

If a rule weight is supposed to stress that a fuzzy rule is more or less "important", this could mean, for example, that the rule is rarely applicable. It could also mean that it is not harmful if the rule is not applied when it should be, or vice versa. But "importance" does not mean that the consequent of a rule should be taken into account only to some extent. This aspect has already been modeled by using fuzzy sets to describe the antecedent.

The interpretation of a rule weight as "reliability" or "trust" is also questionable. If we belief that a rule is reliable only to a certain extent, then this means that we can only trust the final conclusion to a certain extent. However, simply multiplying the output of a rule or its degree of fulfilment by some weight is not an appropriate way to model our belief. For this, probabilistic [Kruse et al., 1991] or possibilistic [Kruse et al., 1994a] methods must be considered.

To view a rule weight as "influence" could mean that a rule with a small weight should only marginally contribute to the system output. Allowing the weights to be selected from $[0, 1]$ could be interpreted as a degree of support for a rule. A value less than 1 would then denote an ill-defined rule that supports its consequent only to some extent.

Often adaptive rule weights are used to tune a fuzzy system such that it produces certain exact output values for certain given input values. Therefore a weight can assume any value in such an approach. In this case it is also possible that negative rule weights occur. If the rule weights are allowed to assume any value in $\mathbb{R}$, some obvious semantical problems occur. It is not clear how rules weighted by values greater than 1 or by negative values should be interpreted.

If the weights are restricted to the interval $[0, 1]$, they can be chosen to reflect the influence of a rule in the computation of the overall output value. But the nature of this influence is obscured. It is preferable to model the influence of a rule by modifying its antecedent such that the rule only applies to a certain degree in a certain situation. In addition one should choose an appropriate consequent that explicitly represents the rule's contribution to the overall output of the fuzzy system.

In some approaches negative rule weights are used to represent "negative rules". In [Halgamuge and Glesner, 1994] it is suggested that we should interpret a rule with a negative weight by "**if not** ... **then** ..." and then use the weight without the negative sign to modify the degree of fulfilment.

However, we have to consider that a fuzzy rule is supposed to represent a vague sample for approximating an otherwise unknown function. The suggested interpretation of rules with negative weights would not have the character of a local sample. It would correspond to a global description of the function:

> **if** the input is **outside** the area specified by the antecedent,
> **then** the function yields the value represented by the conclusion.

By overlapping with other rules, such a "negative rule" would have the effect of an offset or bias for the approximated function.

According to the interpretation of fuzzy rules as vague samples, a rule with a negative weight cannot be viewed as a proposition in the above-mentioned sense. It would only make sense to interpret the rule as a local sample with a function result that is multiplied by a negative value. This means a negative weight could only be applied to the conclusion of a rule.

If the rule weight is to be used to modify the degree of fulfilment, we obtain an invalid negative degree which cannot be properly processed by a fuzzy system. In the special case of a fuzzy classification system a rule with a negative weight might be interpreted as a rule with a negative consequent (**if** ... **then not** class c). Required that the outputs of all rules are combined adequately, a negative weight can have an inhibiting influence on the selection of a class c mentioned in the consequent of the rule. However, we again have the problem that negative degrees of membership may occur, which do not make any sense in a fuzzy system.

From the discussion above we see that the interpretation of rule weights as "importance" or "reliability" are not useful and that negative weights should be avoided by

all means. The interpretation as a kind of influence can be used, if the weights are restricted to $[0, 1]$. However, the influence represented by a rule weight is obscured, because it interferes with the influence expressed by the degree of fulfilment or the consequent.

Depending on whether a rule weight is multiplied to the degree of fulfilment or to the output of a rule, weighting a fuzzy rule is actually equivalent to changing its antecedent or consequent, respectively. Therefore rule weights can always be replaced by changes in the fuzzy sets used in the antecedent and consequent of a rule. As we will show in the following, the implicit modifications of fuzzy sets caused by rule weights can lead to non-normal fuzzy sets and to the fact that identical linguistic values are represented in different ways in different rules.

The Influence of Rule Weights

In the following we study the influence of rule weights on fuzzy systems and concentrate on Mamdani-type fuzzy systems [Mamdani and Assilian, 1975]. In Sugeno-type fuzzy systems [Sugeno, 1985] rule weights have the same impact, if they are applied to the degree of fulfilment. If they are applied to the conclusion, then no semantical problems occur, because the consequents of Sugeno-type rules are functions or constants that are not interpreted linguistically. We briefly discuss Sugeno-type fuzzy systems after discussing the effect of rule weights in Mamdani-type fuzzy systems.

For the sake of simplicity we consider the following two Mamdani-type fuzzy rules to discuss the influence of rule weights:

$$M_1\text{: if } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z \text{ is } D \text{ with } w_1,$$
$$M_2\text{: if } x \text{ is } A \text{ and } y \text{ is } C \text{ then } z \text{ is } D \text{ with } w_2,$$

where A, B, C and D are fuzzy sets of the real line, x and y are input variables, z is the output variable and w_i is a rule weight. Each fuzzy set is labeled with an individual linguistic term. Note that both rules use fuzzy sets A and D.

For Sugeno-type fuzzy systems we consider rules of the form

$$S_1\text{: if } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z = f_1(x, y) \text{ with } w_1,$$
$$S_2\text{: if } x \text{ is } A \text{ and } y \text{ is } C \text{ then } z = f_2(x, y) \text{ with } w_2,$$

where f_i is a function over the input variables and w_i is a rule weight. The overall output for z is computed by a weighted sum (compare Definition 3.4). Note that both rules use fuzzy set A.

Rule Weights Applied to a Degree of Fulfilment

If a positive rule weight is used to modify the degree of fulfilment of a rule we can replace it by changing the membership functions in the antecedents. The following two theorems consider Mamdani-type and Sugeno-type fuzzy systems. For the sake of simplicity we consider only fuzzy systems with one output variable. The extension to multiple output variables is straightforward.

Theorem 5.1 *A Mamdani-type fuzzy system $MF_{\mathcal{R}} : \mathbb{R}^n \to \mathbb{R}$ with a rule base $\mathcal{R}$ of r weighted rules $R_1, \ldots, R_r$ with*

$$R_k : \text{if } x_1 \text{ is } \mu_k^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \mu_k^{(n)} \text{ then } y \text{ is } \nu_k \text{ with } w_k, \ (k = 1, \ldots, r)$$

where the weights $w_k > 0$ are applied to the degrees of fulfilment, is equivalent to a Mamdani-type fuzzy system $MF_{\widehat{\mathcal{R}}} : \mathbb{R}^n \to \mathbb{R}$ with a rule base $\widehat{\mathcal{R}}$ of r unweighted rules $\widehat{R}_1, \ldots, \widehat{R}_r$ with

$$\widehat{R}_k : \text{if } x_1 \text{ is } \widehat{\mu}_k^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \widehat{\mu}_k^{(n)} \text{ then } y \text{ is } \nu_k, \ (k = 1, \ldots, r)$$

such that $\widehat{\mu}_k^{(i)}(x_i) = w_k \cdot \mu_k^{(i)}(x_i)$ $(i = 1, \ldots, n)$.

Proof. *It is sufficient to show that both fuzzy systems compute the same output fuzzy set. For any t-norm $\top$ we obtain*

$$\begin{aligned}
MF_{\mathcal{R}}(\mathbf{x}) &= \text{defuzz}(\nu^*), \text{ with} \\
\nu^*(y) &= \max_{R_k \in \mathcal{R}} \{\top \{\nu_k(y), \ \tau_k\}\} \\
&\underset{(5.1)}{=} \max_{R_k \in \mathcal{R}} \left\{\top \left\{\nu_k(y), \ w_k \cdot \min\{\mu_k^{(1)}(x_1), \ldots, \mu_k^{(n)}(x_n)\}\right\}\right\} \\
&= \max_{R_k \in \mathcal{R}} \left\{\top \left\{\nu_k(y), \ \min\{w_k \cdot \mu_k^{(1)}(x_1), \ldots, w_k \cdot \mu_k^{(n)}(x_n)\}\right\}\right\} \\
&= \max_{\widehat{R}_k \in \widehat{\mathcal{R}}} \left\{\top \left\{\nu_k(y), \ \min\{\widehat{\mu}_k^{(1)}(x_1), \ldots, \widehat{\mu}_k^{(n)}(x_n)\}\right\}\right\} \\
&= \max_{\widehat{R}_k \in \widehat{\mathcal{R}}} \{\top \{\nu_k(y), \ \widehat{\tau}_k\}\} \\
&= \widehat{\nu}^*(y), \text{ where } MF_{\widehat{\mathcal{R}}}(\mathbf{x}) = \text{defuzz}(\widehat{\nu}^*)
\end{aligned}$$

□

Theorem 5.1 is valid for Mamdani-type fuzzy systems that use any t-norm as the inference operator (e.g. the minimum operation as usually or the product). This means that we can replace a positive rule weight by multiplying the individual membership degrees, i.e. we scale the heights of the fuzzy sets of the antecedent with the rule weight. Negative rule weights are not considered. To replace them by

modifying the fuzzy sets, we would also have to replace the mininum operation by the maximum. This means, rules with negative weights can no longer be interpreted as fuzzy rules. The following theorem considers Sugeno-type fuzzy systems.

Theorem 5.2 *A Sugeno-type fuzzy system $SF_{\mathcal{R}} : \mathbb{R}^n \to \mathbb{R}$ with a rule base $\mathcal{R}$ of r weighted rules $R_1, \ldots, R_r$ with*

$$R_k : \text{if } x_1 \text{ is } \mu_k^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \mu_k^{(n)} \text{ then } y = f_k(x_1, \ldots, x_n) \text{ with } w_k, \quad (k = 1, \ldots, r)$$

where the weights $w_k > 0$ are applied to the degrees of fulfilment, is equivalent to a Sugeno-type fuzzy system $SF_{\widehat{\mathcal{R}}} : \mathbb{R}^n \to \mathbb{R}$ with a rule base $\widehat{\mathcal{R}}$ of r unweighted rules $\widehat{R}_1, \ldots, \widehat{R}_r$ with

$$\widehat{R}_k : \text{if } x_1 \text{ is } \widehat{\mu}_k^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \widehat{\mu}_k^{(n)} \text{ then } y = f_k(x_1, \ldots, x_n), \ (k = 1, \ldots, r)$$

such that $\widehat{\mu}_k^{(i)}(x_i) = \sqrt[n]{w_k} \cdot \mu_k^{(i)}(x_i)$ $(i = 1, \ldots, n)$.

Proof.

$$\begin{aligned}
SF_{\mathcal{R}}(\mathbf{x}) &= \frac{\sum_{R_k \in \mathcal{R}} w_k \cdot \tau_k \cdot \ f_k(x_1, \ldots, x_n)}{\sum_{R_k \in \mathcal{R}} w_k \cdot \tau_k} \\
&\underset{(5.1)}{=} \frac{\sum_{R_k \in \mathcal{R}} w_k \cdot \prod_{i=1}^{n} \mu_k^{(i)}(x_i) \ \cdot \ f_k(x_1, \ldots, x_n)}{\sum_{R_k \in \mathcal{R}} w_k \cdot \prod_{i=1}^{n} \mu_k^{(i)}(x_i)} \\
&= \frac{\sum_{R_k \in \mathcal{R}} \prod_{i=1}^{n} (\sqrt[n]{w_k} \ \cdot \ \mu_k^{(i)}(x_i)) \ \cdot \ f_k(x_1, \ldots, x_n)}{\sum_{R_k \in \mathcal{R}} \prod_{i=1}^{n} (\sqrt[n]{w_k} \ \cdot \ \mu_k^{(i)}(x_i))} \\
&= \frac{\sum_{\widehat{R_k} \in \widehat{\mathcal{R}}} \prod_{i=1}^{n} \widehat{\mu}_k^{(i)}(x_i) \ \cdot \ f_k(x_1, \ldots, x_n)}{\sum_{\widehat{R_k} \in \widehat{\mathcal{R}}} \prod_{i=1}^{n} \widehat{\mu}_k^{(i)}(x_i)} \\
&= \frac{\sum_{\widehat{R_k} \in \widehat{\mathcal{R}}} w_k \cdot \widehat{\tau_k} \cdot \ f_k(x_1, \ldots, x_n)}{\sum_{\widehat{R_k} \in \widehat{\mathcal{R}}} w_k \cdot \widehat{\tau_k}} \\
&= SF_{\widehat{\mathcal{R}}}(\mathbf{x})
\end{aligned}$$

□

Theorem 5.2 assumes positive rule weights w_r such that that $\sqrt[r]{w_r}$ is defined. Obviously, the theorem considers only one possible replacement of rule weights. Actually, there are infinite possibilities to distribute a rule weight between the antecedents of a fuzzy rule, where the degree of fulfilment is computed by the product. If negative rule weights are permissible, we can, in addition, decide to modify any odd number of antecedent fuzzy sets by a negative factor leading to a large number of different interpretations.

In every case, a rule weight leads to a modification of the membership degrees of the antecedent fuzzy sets. Therefore, we only consider the most common case where the degree of fulfilment is determined by the minimum. In this case the effect of a rule weight can be neatly demonstrated. Using our two rules M_1 and M_2 from above we obtain the following two modified rules:

$$\begin{array}{ll} M_1^*: & \text{if } x \text{ is } A' \text{ and } y \text{ is } B' \text{ then } z \text{ is } D, \\ M_2^*: & \text{if } x \text{ is } A'' \text{ and } y \text{ is } C'' \text{ then } z \text{ is } D. \end{array}$$

But now for the fuzzy sets in the antecedents we have:

$$\begin{array}{rcl} A', B' & : & \mathbb{R} \longrightarrow [0, w_1], \\ A'', C'' & : & \mathbb{R} \longrightarrow [0, w_2]. \end{array}$$

If we assume $w_1 \neq w_2 \neq 1$, we obtain the following problems:

- Instead of using the same fuzzy set A the rules now use two different fuzzy sets A' and A'' in their antecedents. However, both fuzzy sets are labeled with the same linguistic term, i.e. we now have two different representations for the same linguistic term within the fuzzy system. This is not acceptable if we want to interpret the rule base.

- The resulting fuzzy sets are no longer anymore. The application of the rule weights resulted in rescaling their membership degrees. This also has undesirable effects on the readability of the fuzzy systems. Although the interval $[0, 1]$ is arbitrarily chosen to represent membership degrees, we have to keep in mind that rule weights imply a different interval of membership degrees for each rule.

  Interpretation is affected most, if the weights are larger than 1. In this case, strictly speaking, A', B', A'' and C'' are no longer fuzzy sets.

Figure 5.1 illustrates what happens if two weighted fuzzy rules are replaced by two equivalent unweighted rules with modified antecedent fuzzy sets. In Figure 5.1(a) the two fuzzy rules

if x is A then y is D with 0.5,
if x is A then y is E with 2.0

are displayed. The effect of the rule weights is not visible and we have the impression of two regular, well interpretable fuzzy rules. But if we replace the weighted rules by equivalent unweighted rules with modified antecedent fuzzy sets, we make the influence of the rule weights visible and obtain the situation in Figure 5.1(b). We now see that we really have two fuzzy rules

if x is A' then y is D,
if x is A'' then y is E.

$A' \to [0, 0.5]$ is no longer normal and $A'' \to [0, 2]$ is, strictly speaking, no longer a fuzzy set. However, both A' and A'' still carry the same linguistic label. Figure 5.1(b) illustrates the real impact of rule weights that are applied to the degree of fulfilment and makes clear that a suitable linguistic interpretation of the fuzzy rules is not possible anymore.

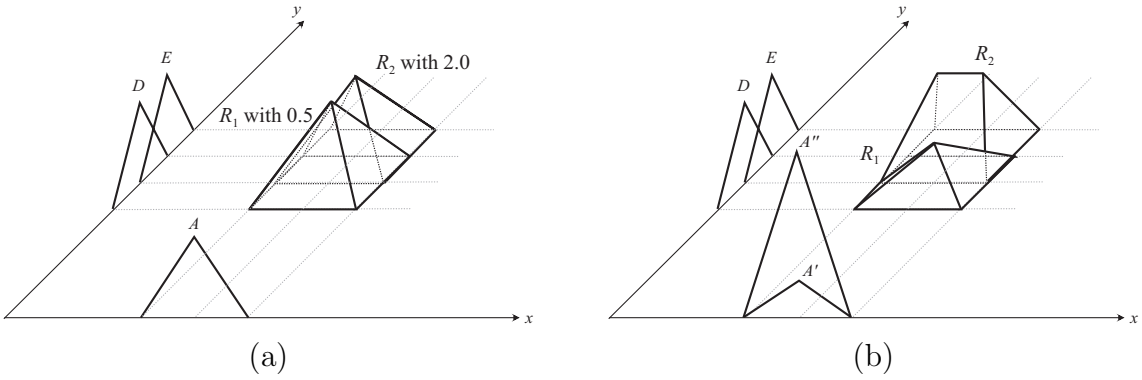


Figure 5.1: The two weighted fuzzy rules of part (a) are replaced in part (b) by equivalent unweighted rules with modified antecedent fuzzy sets

After we have examined the influence of rule weights on the interpretability of the rules we want to discuss their impact on the computation of the output of the rules. In Mamdani-type fuzzy systems rule weights that are used to change the degree of fulfilment of the rules have different effects depending on the selected inference operator.

If we use the minimum to compute the conclusion of a rule, then $w > 1$ only has an effect if $w\tau < 1$. If $w\tau > 1$, the conclusion is equal to the consequent. Weights from $[0, 1]$ result in degrees of fulfilment from $[0, 1]$, and so the computation of the overall output is done as usual.

For a rule with $w < 0$ the conclusion becomes a "negative rectangle" of height $w\tau$ over the whole output domain. After maximum combination with the conclusion fuzzy sets of other possibly active rules, such invalid "negative fuzzy sets" disappear

from the output. Rules with negative weights never contribute to the overall output value, if maximum combination is implemented by the book and takes *all* fuzzy rules into account – even those with zero degree of fulfilment.

However, we have to be careful, if we use an implementation that skips rules with zero activation or areas with zero membership from the maximum combination, for the sake of computation time. In this case, we can obtain a "negative output fuzzy set", or output "fuzzy sets" with negative and positive "membership degrees". The kind of crisp output computed from this depends on the defuzzification method used. If we use a fuzzy software system that allows us to use negative weights, we must have a close look at how max-min-inference and defuzzification are really implemented.

In general, rules with negative weights only influence the computation of the overall output value, if maximum combination is implemented incompletely as described above, or if we use local defuzzification methods, or if we have a fuzzy classifier, where the (weighted) degree of fulfilments are combined directly to indicate a class membership, or if we use a Sugeno-type fuzzy system (see below).

In Figure 5.2 the influence of rule weights on the computation of the output in the case of max-min-inference is shown. Figure 5.2(a) shows rule weights $0 < w_1 < 1$ and $w_2 > 1$. Weight w_1 reduces the degree of fulfilment of the first rule. The product τw_2 is larger than 1, but due to min-implication the output fuzzy set of the second rule is simply the consequent fuzzy set. In Figure 5.2(b) we have a negative weight w_2. This results in an invalid output fuzzy set for the second rule, but the max-combination of the conclusions ignores this result, because the degrees of membership of the output fuzzy set of the first rule are all larger or equal to zero.

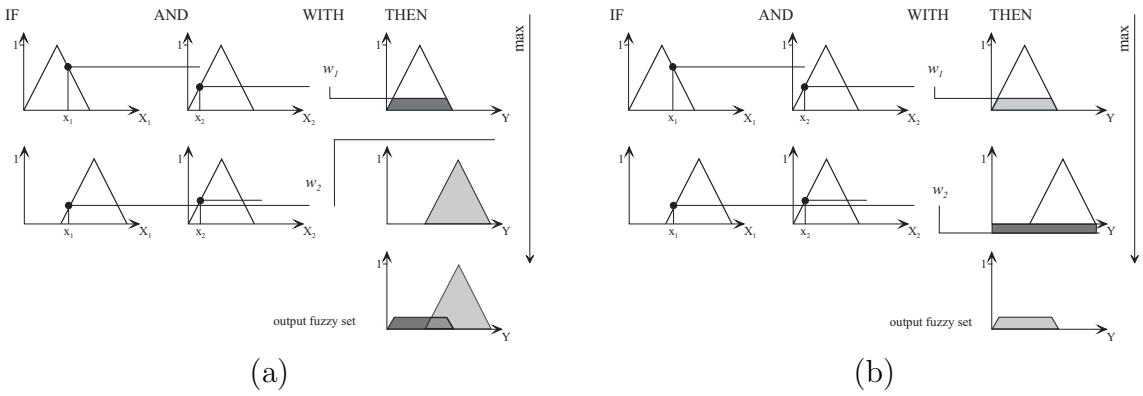


Figure 5.2: The influence of rule weights applied to the degree of fulfilment on the computation of the output in the case of max-min-inference

If we use max-dot-inference and apply the product to compute the conclusion of a rule then the output fuzzy sets are rescaled by the weighted degrees of fulfilment before they are accumulated and defuzzified. Now the case $w\tau > 1$ also influences the computation of the output value. The larger the value that is used to scale the output fuzzy set, the larger is its contribution in the computation of the overall output value if defuzzification procedures like center of gravity are used.

If there are negative rule weights, there are mirrored rescaled "membership functions", which disappear after maximum combination, i.e. the effect is in this case as described above for min-implication and depends on the correct implementation of the max operation. This situation is shown in Figure 5.3, where we have $w_1 < 0$ and $w_2 > 1$.

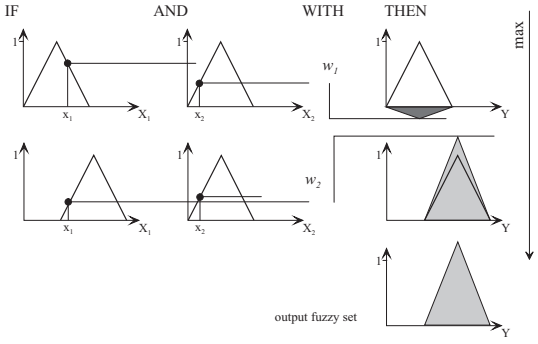


Figure 5.3: The influence of rule weights applied to the degree of fulfilment on the computation of the output in the case of max-dot-inference

In Sugeno-type fuzzy systems the degree of fulfilment is used to compute a weighted sum. Rule weights occur as factors in the sums of the nominator and denominator:

$$z = \frac{\sum_i \tau_i \cdot w_i \cdot f_i(x, y)}{\sum_i \tau_i \cdot w_i}.$$

We obtain a different result from that when we apply weights to the conclusions, where the weights would only occur in the nominator.

Weighted Sugeno-type rules can be replaced by equivalent unweighted rules, if we change the antecedent fuzzy sets accordingly. This means rule weights applied to the degree of fulfilment cause the same semantical problems in Sugeno-type fuzzy systems as they do in Mamdani-type fuzzy systems.

Rule Weights Applied to Conclusions

If a rule weight is applied to the conclusion of a rule, it modifies the size of a rule's output value. If the output is crisp, the weight is simply multiplied with this value. If an output fuzzy set is multiplied with a weight, its support and shape are modified.

Theorem 5.3 *A Mamdani-type fuzzy system $MF_{\mathcal{R}} : \mathbb{R}^n \to \mathbb{R}$ with a rule base $\mathcal{R}$ of r weighted rules $R_1, \ldots, R_r$ with*

$$R_k : \text{if } x_1 \text{ is } \mu_k^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \mu_k^{(n)} \text{ then } y \text{ is } \nu_k \text{ with } w_k, \ (k = 1, \ldots, r)$$

where the weights are applied to the conclusions, is equivalent to a Mamdani-type fuzzy system $MF_{\widehat{\mathcal{R}}} : \mathbb{R}^n \to \mathbb{R}$ with a rule base of r unweighted rules $\widehat{R}_1, \ldots, \widehat{R}_r$ with

$$\widehat{R}_k : \text{if } x_1 \text{ is } \mu_k^{(1)} \text{ and } \ldots \text{ and } x_n \text{ is } \mu_k^{(n)} \text{ then } y \text{ is } \widehat{\nu}_k, \ (k = 1, \ldots, r)$$

such that

$$\widehat{\nu}_k = w_k \cdot \nu_k \text{ with } \widehat{\nu}_k(y) = (w_k \cdot \nu_k)(y) = \sup\{\nu_k(t) | y = w_k \cdot t\} \ (k = 1, \ldots, r).$$

Proof. *For any t-norm $\top$ we obtain*

$$\begin{aligned} MF_{\mathcal{R}}(\mathbf{x}) &= \text{defuzz}(\nu^*), \text{ with} \\ \nu^*(y) &= \max_{R_k \in \mathcal{R}} \{\top\{(w_k \cdot \nu_k)(y), \ \tau_k\}\} \\ &= \max_{\widehat{R}_k \in \mathcal{R}} \{\top\{\widehat{\nu}_k(y), \ \tau_k\}\} \\ &= \widehat{\nu}^*(y), \text{ where } MF_{\widehat{\mathcal{R}}}(\mathbf{x}) = \text{defuzz}(\widehat{\nu}^*) \end{aligned}$$

□

This means we can replace the weights in our original rules M_1 and M_2 by changing the membership functions in the consequents:

$$\begin{aligned} M_1^*&: \text{ if } x \text{ is } A \text{ and } y \text{ is } B \text{ then } z \text{ is } D', \\ M_2^*&: \text{ if } x \text{ is } A \text{ and } y \text{ is } C \text{ then } z \text{ is } D''. \end{aligned}$$

Let us assume that the support of the original (convex) output fuzzy set D is $[l, u]$. Then we obtain for the support of D' and D'' $[lw_1, uw_1]$ and $[lw_2, uw_2]$, respectively, or for negative weights $[uw_1, lw_1]$ and $[uw_2, lw_2]$, respectively. This results in the following problems:

- Instead of a single fuzzy set D two different fuzzy sets D' and D'' are now used in our two rules. However, they are labeled with the same linguistic term, i.e. we obtain two different representations of the same linguistic value in our fuzzy system.

- The membership functions of the consequents are shifted away from their original positions, and their supports are rescaled. On the one hand, this can result in undesirable small or large supports and, on the other hand, a fuzzy set can even migrate from a positive part of its domain to a negative part and vice versa. It is also possible that a fuzzy set completely leaves the domain of the output variable. If we are interested in interpreting the fuzzy systems we are forced to relabel the fuzzy sets.

Figure 5.4 illustrates how two fuzzy rules with weights that are applied to the output fuzzy sets can be replaced by equivalent unweighted rules. In Figure 5.4(a) we can see the two fuzzy rules

$$\begin{array}{l}\text{if } x \text{ is } A \text{ then } y \text{ is } D \text{ with } 0.5,\\ \text{if } x \text{ is } B \text{ then } y \text{ is } D \text{ with } 2.0.\end{array}$$

If we replace the weighted rules by equivalent unweighted rules, we obtain the scenario in Figure 5.4(b), where we have the rules

$$\begin{array}{l}\text{if } x \text{ is } A \text{ then } y \text{ is } D',\\ \text{if } x \text{ is } B \text{ then } y \text{ is } D''.\end{array}$$

We can see that D' and D'' are scaled versions of D, but remain normal convex fuzzy sets. However, both fuzzy sets still carry the same linguistic label and to insure the interpretability of the fuzzy system, we must relabel them.

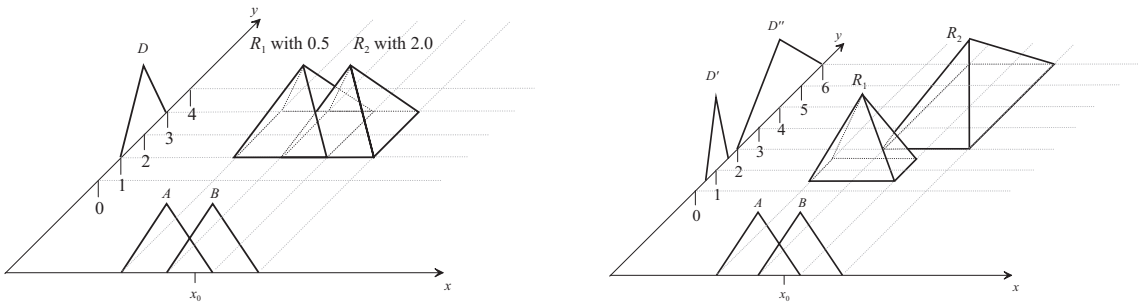


Figure 5.4: The two weighted fuzzy rules of part (a) are replaced in part (b) by equivalent unweighted rules with modified consequent fuzzy sets

Replacing the weighted rules by equivalent unweighted rules shows which part of the data space is really covered by the fuzzy rules. In this sense rule weights that are applied to conclusions obscure the interpretation of a fuzzy rule base.

In contrast to the application of a rule weight to a degree of fulfilment we do not encounter problems in computing the overall output value if we apply a weight to the conclusion, because non-normal or invalid fuzzy sets do not occur. Even negative weights or weights larger than 1 only result in changing the support of the output fuzzy sets. So the problems of these kinds of rule weights are multiple representations of the same linguistic term and the "invisible" effects on the location and form of the output fuzzy sets.

If a rule weight is used to modify the conclusion of a rule in a Sugeno-type fuzzy system, then the weights appear as additional factors in the nominator of the equation for the overall output value:

$$SF_{\mathcal{R}}(\mathbf{x}) = \frac{\sum\limits_{R_k \in \mathcal{R}} \tau_k \cdot w_k \cdot f_k(\mathbf{x})}{\sum\limits_{R_k \in \mathcal{R}} \tau_k}.$$

Applying rule weights to conclusions is equivalent to replacing the function f_k, which computes the output of a rule R_k by $f'_k = w_k f_k$. In this case we do not encounter semantical problems, because the consequents of Sugeno-type rules are usually not interpreted linguistically. This means for Sugeno-type fuzzy systems rule weights that are applied to the conclusions are a simple way to train the system in a linear fashion. Such an approach was, for example, successfully applied in a neuro-fuzzy system for stock index prediction [Siekmann et al., 1999].

Avoiding Rule Weights

We have shown how weighted fuzzy rules can be replaced by equivalent unweighted fuzzy rules. If this is done the effects of rule weights on a fuzzy system become visible. The following list summarizes these problems:

(i) Multiple representations of the same linguistic term.

(ii) Non-normal or invalid fuzzy sets, if the weight is applied to the degree of fulfilment.

(iii) Effects on the computation of the output value that are not visible in the representation of weighted fuzzy rules.

(iv) Difficult or even impossible linguistic interpretation of the rule base.

Rule weights are a very simple way to obtain adaptability for a fuzzy system. The performance can be easily tuned by applying a learning algorithm to the rule weights. Because all other parameters of the fuzzy system remain constant, a linear learning algorithm like the delta rule [Widrow and Hoff, 1960] or a least mean square procedure can be applied. This is much simpler to implement than to specify learning algorithms that modify fuzzy sets directly.

In many cases, rule weights alone are not sufficient to enhance the performance of a fuzzy system by learning, because rule weights cannot change the positions of the antecedent fuzzy sets. If this is necessary, then the parameters of the membership functions must also be adapted by a learning procedure.

Some of the problems of rule weights can be avoided, if negative weights are not used and the rule weights are normalized. In this case membership degrees that are not from $[0, 1]$ do not occur. A rule weight from $[0, 1]$ that is applied to the degree of fulfilment can be viewed as a *reduction of influence* of a rule. In this case the rule weight has a similar interpretation as the degree of fulfilment of a rule. The difference is that the degree of fulfilment reflects the (posterior) influence of the rule due to the current input data and the weight would reflect the (prior) influence specified by an expert or obtained during a learning process. Negative rule weights or weights larger than 1 do not make any sense in this interpretation and must be avoided.

But even if we choose rule weights from $[0, 1]$ that can be interpreted to some extent, we must be aware that weighted rules can always be replaced by equivalent unweighted rules with modified fuzzy sets. Unweighted rules are preferable because the effects of the weights on a fuzzy system are not visible. Users may tend to forget these effects – especially the effect of multiple representations of identical linguistic terms – when they interpret the rule base linguistically.

Rule weights that are applied to the conclusions have no interpretation. They simply scale the output fuzzy sets and are just a hidden modification of the consequents. Normalizing such rule weights makes no sense because they are used to change the conclusions in order to influence the size of the overall output value of the fuzzy system.

We can view a fuzzy system as a (fuzzy) combination of local models, which are fuzzy sets for Mamdani-type systems, and (usually) linear models for Sugeno-type systems. If a rule weight is applied to the degree of fulfilment then the influence of the local model represented by the consequent is changed for a certain area of the data space. If a rule weight is applied to the conclusion of a rule, then the represented local model itself is changed. For the sake of interpretation these changes should not be done by using rule weights, but instead by changing the respective membership functions directly.

If we provide a learning algorithm that modifies the parameters of the membership functions, it is also easily possible to define restrictions for the learning process. These restrictions can make sure that certain changes to the fuzzy sets must not be carried out to ensure, for example, that the fuzzy sets stay within their domains, keep their relative positions etc. By restricting the learning algorithms we sacrifice degrees of freedom in exchange for readability. Learning becomes more difficult and for an increase in readability we pay with a loss of accuracy [Bersini et al., 1998, Nauck et al., 1997]. In the following sections of this chapter we discuss several learning algorithms that directly modify parameters of membership functions.

5.2 Training Membership Functions

We have seen in the previous section that adaptive rule weights in fuzzy systems can prevent a suitable linguistic interpretation. To train a fuzzy system it is preferable to specify a learning procedure that is able to modify fuzzy sets directly.

Many approaches to training membership functions are based on backpropagation implemented by gradient descent [Nomura et al., 1992] and are sometimes also mixed with adaptive rule weights, like, for example, in the ARIC [Berenji, 1992] or FuNe [Halgamuge and Glesner, 1994, Halgamuge, 1995] models.

In order to use gradient descent techniques, the membership functions and all functions that take part in the evaluation of the fuzzy rule base must be differentiable [Grauel and Ludwig, 1999]. If we use a regular Mamdani-type fuzzy system with max-min-inference this is not the case. There are also problems, when non-continuous membership functions like triangular or trapezoidal functions or non-differentiable defuzzification functions are used.

In Sugeno-type fuzzy systems gradient descent learning can be more easily applied, because they use only the product as t-norm and compute the output by a weighted sum. If bell-shaped (Gaussian) membership functions are used, then a Sugeno-type fuzzy system is equivalent to a special kind of RBFN (Definition 3.10) [Jang and Sun, 1993]. In this case the fuzzy system uses only differentiable functions, and a standard backpropagation algorithm based on gradient descent can be applied to the parameters of the fuzzy sets.

Nomura et al. describe a learning procedure for simplified Sugeno-type fuzzy systems that only use constants in the consequents of the rules [Nomura et al., 1992]. The parameter updates of a gradient-descent-based supervised learning procedure can easily be computed by applying the chain-rule. However, in [Nomura et al., 1992] the fuzzy sets in the antecedents are represented by triangular membership functions and the authors ignored the fact that each membership function cannot be differentiated at three points. A simple heuristic to cope with this problem is to not update a parameter, if its value occurs in the input [Nauck et al., 1997].

Nomura et al. also allowed the fuzzy sets of each rule to be modified independently from all other rules. This results in multiple fuzzy set representations of the same linguistic term. If the antecedents are to be interpreted linguistically after learning, then this approach is not acceptable. Refinements of the learning procedure that ensure that each linguistic term is represented by just one fuzzy set were suggested in [Bersini et al., 1993, Nauck, 1994b, Nauck et al., 1997].

The ANFIS model [Jang, 1993] implements a regular Sugeno-type fuzzy system with continuous bell-shaped membership functions. The consequents consist of linear combinations of the input variables. In the training procedure of ANFIS each learning step has two parts. In the first part the parameters of the antecedent fuzzy sets are assumed to be constant and after the current input has been propagated the optimal consequent parameters are computed by least squares estimation. In the second part the newly computed consequent parameters are assumed to be constant and after the same input has been propagated again the antecedent parameters are updated once by gradient descent. This procedure is iterated until the overall error is small enough or does not change anymore. Jang reports that this two-stage approach performs better than simply using gradient descent to compute all parameter updates [Jang, 1993]. However, with an increasing number of consequent parameters the ANFIS learning algorithm becomes very expensive. A comparsion between ANFIS and NEFPROX – which uses fast heuristics instead of gradient descent and LSE – can be found in [Nauck and Kruse, 1999a].

To be able to apply a gradient-descent-based learning algorithm to a Mamdani-type fuzzy system some neuro-fuzzy approaches replace the usual max-min inference by some other evaluation procedure. In the GARIC model [Berenji and Khedkar, 1992, Berenji, 1998], for example, the degree of fulfilment of a fuzzy rule is computed by a so-called *soft minimum* function $\widetilde{\min}$ (5.3), which can be differentiated and converges to the minimum operation if its parameter α goes to infinity.

$$\widetilde{\min}(x_1,\ldots,x_n) = \frac{\sum_{i=1}^{n} x_i \mathrm{e}^{-\alpha x_i}}{\sum_{i=1}^{n} \mathrm{e}^{-\alpha x_i}} \tag{5.3}$$

Instead of using max-min inference and defuzzification the GARIC model uses the degrees of fulfilment to directly obtain a crisp output value for each rule by a (differentiable) *local defuzzification* function. The output values of the rules are then combined via a weighted sum to the overall output value of the fuzzy system. This evaluation procedure uses only differentiable functions and therefore a gradient-descent-based learning procedure can be defined. In GARIC triangular membership functions are used. The problem of non-differentiable points is solved by using the mean of the differentiation from the left and from the right at these points [Berenji and Khedkar, 1992].

If the evaluation procedure in Mamdani-type fuzzy systems is not to be modified in order to apply a gradient descent, then other means to adapt the membership functions must be used. Recent approaches choose to implement backpropagation by simple heuristics instead of gradient descent [Nauck and Kruse, 1994, Nauck and Kruse, 1995, Nauck and Kruse, 1997c]. The general idea of such heuristics is to determine whether a fuzzy rule should contribute more or less to the overall output of a fuzzy system. This information can then be used to slightly modify the membership function in a suitable way after which this process is iterated.

The output error can be, for example, computed by the sum of squared errors SSE (Eq. 3.1). For the application of fuzzy systems an exact output value is often not required and approximate solutions are acceptable. In this case other error measures can be used that tolerate certain deviations from the target values. The neuro-fuzzy control approach NEFCON [Nauck, 1994a, Nürnberger et al., 1999], for example, uses a fuzzy error measure that is computed by a fuzzy rule base. In control applications an error value cannot be computed directly, because the target values are not known. The training procedure must therefore use reinforcement learning [Kaelbling et al., 1996] – a special case of supervised learning, where the error represents a failure or success of the trained system and not the distance to a target output.

If the target value for a given input pattern is known, a fuzzy error can be defined by specifying a fuzzy set that represents the linguistic term *the output is correct*. This can be done, for example, by

$$C : \mathbb{R} \to [0,1], \quad C(d) = \mathrm{e}^{-\left(\frac{a \cdot d}{d_{\max}}\right)^2}, \tag{5.4}$$

with $d = t - o$, where t is a target value and o is an output value. The parameter $d_{\max}$ specifies the (absolute) difference from where the degree of membership is smaller than e^{-a^2}. For a values like 1.7 ($\mathrm{e}^{1.7^2} \approx 0.05$) or 2.3 ($\mathrm{e}^{2.3^2} \approx 0.01$) are suitable. Other forms of membership functions are also possible, for example, triangular or trapezoidal forms.

From the degree to which an output is correct, we can easily derive the error value

$$\mathrm{FE} : \mathbb{R} \to [-1,1], \quad \mathrm{FE}(d) = \mathrm{sgn}(d) \cdot (1 - C(d)). \tag{5.5}$$

We call this error measure *fuzzy error*, because it is based on a fuzzy set that models the concept *correct*. For more complex learning situations like in NEFCON the concept *correct* is modelled by a fuzzy rule base. The overall performance of the fuzzy system can be measured by the sum of absolute fuzzy errors

$$\mathrm{SAFE} = \sum_{p \in \tilde{\mathcal{L}}} |\mathrm{FE}^{(p)}| = \sum_{p \in \tilde{\mathcal{L}}} \sum_{j=1}^{m} |\mathrm{FE}_j^{(p)}| = \sum_{p \in \tilde{\mathcal{L}}} \sum_{j=1}^{m} (1 - C(t_j^{(p)} - o_j^{(p)})) \tag{5.6}$$

that assumes values between 0 (all outputs are correct) and $s = \left|\tilde{\mathcal{L}}\right|$ (all outputs are not correct).

Like SE the error measure FE tolerates small errors, but there is virtually no distinction between large and very large errors. Because the error is bounded by -1 and 1, outliers do not have such a strong influence on the training procedure as it is the case if SE is used, for example, where the error is not bounded and large errors ($t - o \gg 1$) tend to be overemphasized.

There is another advantage of FE. If there are several output variables, we can specify an individual error for each by specifying different values for $d_{\max}$. Thus it is not necessary to normalize the training data as is usually done for neural networks to avoid large differences in the size of error signals for output variables on different scales.

The error measure is not only used to guide the learning process, but also to evaluate the performance of the final model. The error on the training set is not useful for this, because a small error can point to overfitting. To compute the error on a test set not used for training is also usually not sufficient, because the way the data was selected may influence the performance. In order to reduce these influences a resampling technique called *cross validation* can be used. This is also useful, if only little training data is available and a large test set cannot be afforded. In n-fold cross validation the training data is split randomly into n stratified samples. Then the model is generated n times, each time using $n - 1$ samples and one sample for testing such that each sample is used once for testing. The final model is created by using all n samples, i.e. all available data If n is equal to the number of training patterns, we speak of 1-*leave-out cross validation*.

The error estimation (mean error) $\bar{e}$ for unseen data is computed from the errors e_i of the classifiers created during validation. In order to evaluate the reliabiliy of the error value the variance of the error distribution should be considered and a confidence interval should be computed. The 99% confidence interval for the estimated error is given by

$$\bar{e} \pm 2.58 \cdot \hat{\sigma}_{\bar{e}}$$

where $\hat{\sigma}_{\bar{e}}$ is the standard error of the mean:

$$\hat{\sigma}_{\bar{e}} = \sqrt{\frac{\sum_{i=1}^{n}(e_i - \bar{e})^2}{n \cdot (n-1)}}. \tag{5.7}$$

5.3 Mamdani-type Fuzzy Systems

In this section we discuss a learning algorithm applied to Mamdani-type fuzzy systems for function approximation. The supervised learning algorithm that is presented below is based on backpropagation. An error is determined at the output side of the fuzzy system and is propagated backwards through the architecture. The main information of the error value is whether the contribution of a fuzzy rule to the overall output value should be increased or decreased. The size of the error value is used to compute the size of the modification.

The learning algorithm is presented in four parts in Algorithms 5.1 – 5.4. Algorithm 5.1 implements the main loop of the training procedure. In each loop the algorithm propagates a training pattern, determines the output of the fuzzy system and computes the parameter updates of the consequent and of the antecedent membership functions. Depending on whether online learning or batch learning is selected, the fuzzy sets are either updated immediately after each pattern, or only after all training patterns have been presented once, i.e. after a complete epoch.

The algorithms use the following notations:

- $\tilde{\mathcal{L}}$: a set of training data (fixed learning problem) with $\left|\tilde{\mathcal{L}}\right| = s$, which represents a function approximation problem, where patterns $\mathbf{p} \in \mathbb{R}^n$ must be mapped to target vectors $\mathbf{t} \in \mathbb{R}^m$.
- $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$: a training pattern consists of an input vector $\mathbf{p} \in \mathbb{R}^n$ (input pattern) and a target vector $\mathbf{t} \in \mathbb{R}^m$.
- $A_r = (\mu_r^{(1)}, \ldots, \mu_r^{(n)})$: the antecedent of rule R_r. $A_r(\mathbf{p})$ denotes the degree of fulfilment of rule R_r (with antecedent A_r) for pattern $\mathbf{p}$, i.e. $A_r(\mathbf{p}) = \min\{\mu_r^{(1)}(p_1), \ldots, \mu_r^{(n)}(p_n)\}$.
- $\mu_r^{(i)}$: a fuzzy set of input variable $x_i (i \in \{1, \ldots, n\})$ that appears in the antecedent of fuzzy rule $R_r (r \in \{1, \ldots, k\})$.
- $\nu_r^{(j)}$: a fuzzy set of output variable $y_j (j \in \{1, \ldots, m\})$ that appears in the consequent of fuzzy rule R_r.

Algorithm 5.1 also uses the following options:

- MINFSONLY: If this option is selected, then in each rule only the antecedent fuzzy set is modified that yields the smallest degree of membership of all fuzzy sets in the antecedent of the considered rule. Otherwise, all fuzzy sets in the antecedent are modified.
- ONLINELEARNING: If this option is selected, then the fuzzy sets are modified after each propagation of a training pattern.

- BATCHLEARNING: If this option is selected, then the computed updates for the fuzzy sets are accumulated first. The fuzzy sets are not modified before all training patterns have been propagated, i.e. after a complete epoch.

The end criterion that terminates Algorithm 5.1 can depend on any combination of the following conditions:

- the overall output error is small enough,
- the overall output error has reached a local minimum,
- a maximum number of iterations has been exceeded.

It is useful to observe these conditions on a separate validation set and not on the training set in order to avoid over-generalization.

When the antecedent of a rule R_r is updated, we can choose between the modification of all fuzzy sets used in the antecedent or just of that fuzzy set $\mu_r^{\min}$ which yields the minimum degree of membership:

$$\mu_r^{\min} = \mu_r^{(j)} \text{ with } \mu_r^{(j)}(x_j) = \tau_r = \mu_r(\mathbf{x}) = \min\{\mu_r^{(1)}(x_1), \ldots, \mu_r^{(n)}(x_n)\} \tag{5.8}$$

Updating only $\mu_r^{\min}$ is motivated by the idea to keep the changes to the fuzzy systems as small as possible. To change the degree of fulfilment of a rule it is sufficient to change $\mu_r^{\min}$. If $\mu_r^{\min}$ is not unique, then ties are either broken arbitrarily or all fuzzy sets with this property are modified.

Figure 5.5 shows the difference between both approaches for an antecedent of a rule using two input variables x_1 and x_2. The fuzzy set $\mu : x_1 \times x_2 \to [0,1],\ \mu(x_1, x_2) = \min(\mu^{(1)}(x_1), \mu^{(2)}(x_2)$, which represents the rule antecedent has the form of a pyramid and is seen from above. Assume we present a training pattern as seen in Figure 5.5a. The degree of fulfilment of the rule is equal to the degree of membership of $\mu^{(1)}(x_1)$. Let us further assume the degree of fulfilment of the rule must be increased. In this case it is sufficient to modify $\mu^{(1)}$ (Figure 5.5b). Only if the goal of the training procedure is to move the core of the antecedent closer to the training pattern in both dimensions, must both fuzzy sets be modified (Figure 5.5c).

The algorithms for computing the updates of the membership functions in the consequent and the antecedent of a rule are simple heuristics (Algorithms 5.2 and 5.3). The computations are given such that the membership functions stay symmetrical. If this is not required, only that section of a membership function needs to be modified, where the current input value is located. Bell-shaped membership functions are always symmetrical. Depending on whether the fuzzy set to be modified is used in an antecedent or in a consequent of a rule one of the following two strategies is used.

Algorithm 5.1: Fuzzy set learning in a Mamdani-type fuzzy system

repeat
  for all patterns $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$ **do** *(* there are s training patterns *)*
    propagate the next training pattern $(\mathbf{p}, \mathbf{t})$;
    for all output variables y_j **do**
      compute the output error E_j; *(* based on $t_j - o_j$, see page 96 *)*
    end for
    for each rule R_r with $A_r(\mathbf{p}) > 0$ **do**
      for all $\nu_r^{(j)}$ **do** *(* modify all output fuzzy sets *)*
        ComputeConsequentUpdates$(\nu_r^{(j)}, E_j, A_r(\mathbf{p}), t_j)$; *(* see Alg. 5.2 *)*
        if (ONLINELEARNING) **then** *(* update after each pattern *)*
          Update $(\nu_r^{(j)}, 1)$; *(* see Algorithm 5.4 *)*
        end if
      end for
      $\mathcal{E}_r = A_r(\mathbf{p}) \cdot (1 - A_r(\mathbf{p})) \cdot \frac{1}{m} \sum_{y_j} (2 \cdot \nu_r^{(j)}(t_j) - 1) \cdot |E_j|$; *(* see Eq. (5.10) *)*
      if (MINFSONLY) **then** *(* modify only one antecedent fuzzy set *)*
        $j = \underset{i \in \{1,\ldots,n\}}{\operatorname{argmin}} \{\mu_r^{(i)}(p_i)\}$;
        ComputeAntecedentUpdates$(\mu_r^{(j)}, \mathcal{E}_r, p_j)$; *(* see Algorithm 5.3 *)*
        if (ONLINELEARNING) **then** *(* update after each pattern *)*
          Update $(\mu_r^{(j)}, 1)$; *(* see Algorithm 5.4 *)*
        end if
      else *(* modify all antecedent fuzzy sets *)*
        for all $\mu_r^{(i)}$ **do**
          ComputeAntecedentUpdates$(\mu_r^{(i)}, \mathcal{E}_r, p_i)$; *(* see Algorithm 5.3 *)*
          if (ONLINELEARNING) **then** *(* update after each pattern *)*
            Update $(\mu_r^{(i)}, 1)$; *(* see Algorithm 5.4 *)*
          end if
        end for
      end if
    end for *(* end for each rule *)*
  end for *(* end for all training patterns *)*
  if (BATCHLEARNING) **then** *(* update after complete epoch *)*
    for all output variables y_i **do**
      for all fuzzy sets $\nu_j^{(i)}$ **do**
        Update $(\nu_j^{(i)}, s)$; *(* see Algorithm 5.4 *)*
      end for
    end for
    for all input variables x_i **do**
      for all fuzzy sets $\mu_j^{(i)}$ **do**
        Update $(\mu_j^{(i)}, s)$; *(* see Algorithm 5.4 *)*
      end for
    end for
  end if
until end criterion;

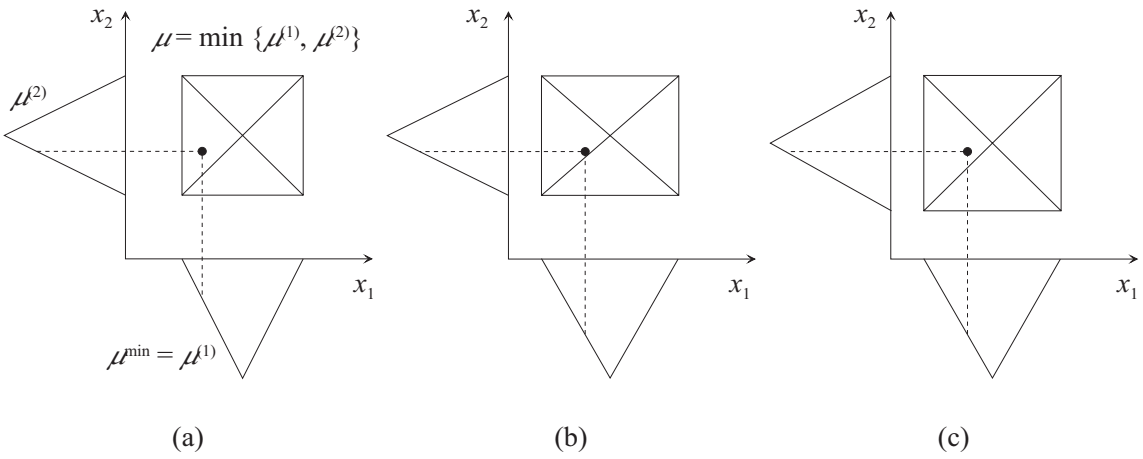


Figure 5.5: An antecedent of two variables in the product space $x_1 \times x_2$ (a). To increase the degree of fulfilment it is sufficient to modify only fuzzy set $\mu^{\min} = \mu^{(1)}$ (b). Both fuzzy sets are changed, if the core of the antecedent is to be moved towards the training pattern (c).

- **Antecedent**: If the degree of membership must be increased, the support of the fuzzy set is enlarged and shifted such that the core of the fuzzy sets is moved towards the current input value. If the degree of membership must be decreased the opposite modification is carried out: the support is reduced and shifted such that the core is moved away from the current input value. An example for the modification of a triangular membership function is shown in Figure 5.6.

- **Consequent**: A heuristics for modifying consequent fuzzy sets must take the defuzzification procedure into account. Usually a defuzzification strategy that computes a weighted average is used, for example, center of gravity or mean of maximum. To move the output value of a fuzzy system closer to the target value the support of a consequent fuzzy set must shifted such that the core of the fuzzy sets moves closer to the target value. If the target has non-zero membership with the fuzzy set to be modified, then the support of this fuzzy set is also reduced to focus the fuzzy set on the target value. If the target has zero membership with a fuzzy set to be modified, then the support of this fuzzy set is extended towards the target value. An example for the modification of two triangular membership functions is shown in Figure 5.7.

The modifications of the consequent fuzzy sets are computed by using the output error of the corresponding variable. In order to compute updates for an antecedent fuzzy set, we need to know the error of a fuzzy rule. If we assume the neural network view of a fuzzy system, we can compute a suitable error value by backpropagating the output error from the output nodes to the rule nodes. To correct the output of a

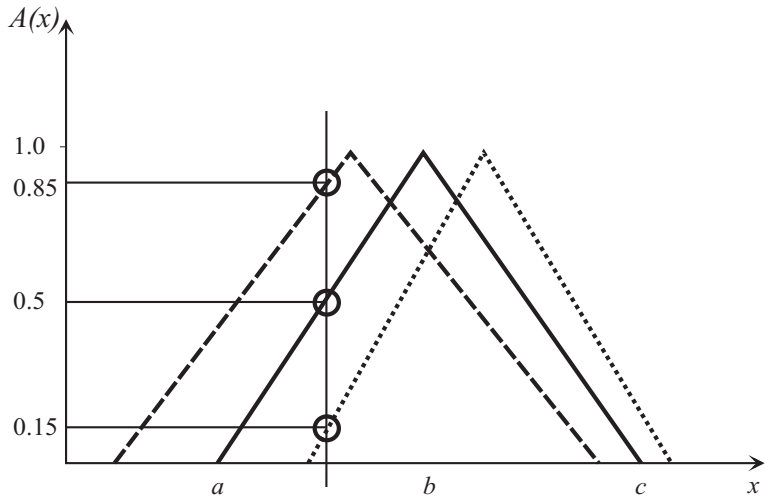


Figure 5.6: To increase the degree of membership for the current input pattern the original representation of the fuzzy set (center) assumes the representation on the right, to decrease the degree of membership, it assumes the representation on the left

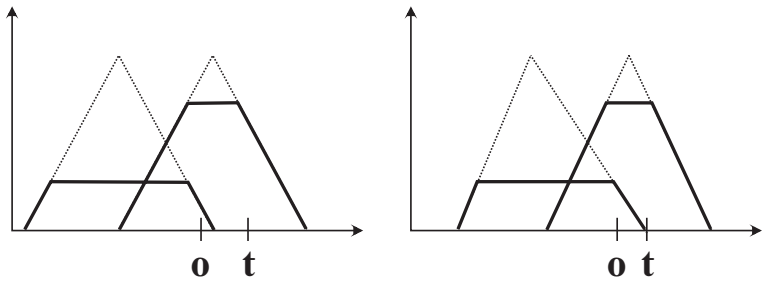


Figure 5.7: To move an output value o of a fuzzy system closer to a current target value t the consequent fuzzy sets are moved towards t

fuzzy system by modifying the degrees of fulfilment of the rules we must increase the influence of rules where a consequent fuzzy set yields a high degree of membership for the target value. If the target value has only a small degree of membership with the consequent fuzzy set, the influence of the corresponding rule must be decreased. A fuzzy rule R_r obtains the following error signal from its output variables $y_1, \ldots, y_m$:

$$\frac{1}{m} \sum_{j=1}^{m} 2\nu_r^{(j)}(t_j) - 1|E_j| \tag{5.9}$$

For $\nu_r^{(j)}(t_j) > 0.5$ we obtain a positive error signal from y_j in order to increase the degree of fulfilment. For $\nu_r^{(j)}(t_j) < 0.5$ the error signal from y_j is negative in order to decrease the degree of fulfilment of R_r. For $\nu_r^{(j)}(t_j) = 0.5$ it cannot be decided

if it would be better to increase or decrease the influence of R_r given the current target t_j and therefore the error signal from y_j is zero.

The rule error must also depend on the current degree of fulfilment τ_r of a rule. Because a fuzzy system is a combination of local models, we can try to reach a state where a fuzzy rule either provides a very high or a very low degree of fulfilment for all training patterns. This means, the partitioning of the data space by the fuzzy rules should be as crisp as possible. The final rule error is therefore computed by

$$\mathcal{E}_r = \tau_r(1-\tau_r)\frac{1}{m}\sum_{j=1}^{m}(2\nu_r^{(j)}(t_j)-1)|E_j|. \tag{5.10}$$

The rule error is zero for $\tau_r = 0$ and $\tau_r = 1$. A rule with $\tau_r = 0$ does not contribute to the current output and should not be trained. A rule with $\tau_r = 1$ perfectly matches the current input pattern. In this case we assume the the consequent alone is responsible for the error and that the antecedent must not be changed. The rule error is maximum for $\tau_r = 0.5$ in order to force a rule to "decide" to either provide large or small degrees of fulfilment.

To compute the parameter updates of a fuzzy set, we must take the form of the membership function, i.e. the meaning of the parameters, into account. The procedure *ComputeUpdates* (Algorithm 5.4) provides the necessary computations for triangular (5.11), trapezoidal (5.12) and bell-shaped membership functions (5.13) as given by the following equations:

$$\mu_{a,b,c} : \mathbb{R} \to [0,1], \quad \mu_{a,b,c}(x) = \begin{cases} \frac{x-a}{b-a} & \text{if } x \in [a,b), \\ \frac{c-x}{c-b} & \text{if } x \in [b,c], \\ 0 & \text{otherwise,} \end{cases} \tag{5.11}$$

with $a \leq b \leq c$,

$$\mu_{a,b,c,d} : \mathbb{R} \to [0,1], \quad \mu_{a,b,c,d}(x) = \begin{cases} \frac{x-a}{b-a} & \text{if } x \in [a,b), \\ 1 & \text{if } x \in [b,c], \\ \frac{d-x}{d-c} & \text{if } x \in (c,d], \\ 0 & \text{otherwise,} \end{cases} \tag{5.12}$$

with $a \leq b \leq c \leq d$,

$$\mu_{a,b,c} : \mathbb{R} \to [0,1], \quad \mu_{a,b,c}(x) = \mathrm{e}^{\left(\frac{c\cdot(x-b)}{a}\right)^2} \tag{5.13}$$

with $a > 0, c > 0, b \in \mathbb{R}$.

A triangular membership function (5.11), which can be used to represent fuzzy numbers (approximately b), is given by its center b and its support $[a, c]$. A trapezoidal

membership function (5.12) can be used to represent a fuzzy interval (approximately between b and c). It is given by its core $[b, c]$ and its support $[a, d]$. A bell-shaped membership function (5.13) is also useful to represent fuzzy numbers and can be used, for example, if a differentiable function is needed, or if the support must cover the whole domain. The parameter b specifies the center of the function and a specifies the half width of the α-cut at e^{-c^2}. Commonly $c = 1.7$ or $c = 2.2$ is chosen such that a specifies the half width of the α-cut at $\mathrm{e}^{-2.89} \approx 0.05$ or $\mathrm{e}^{-4.84} \approx 0.01$.

To ensure an acceptable interpretability of the fuzzy rule base after learning, the training algorithm must not apply arbitrary updates to the fuzzy sets. Before the updates computed by Algorithms 5.2 and 5.3 are applied we must check whether the resulting forms of the membership functions are acceptable. We usually specify a number of constraints on the parameters before learning which must be satisfied by the training algorithm.

We can separate the parameters of a membership function into position parameters and width parameters, which determine the location and size of the support, respectively. For example, in a bell-shaped membership function (5.13) b is a position parameter and a is a width parameter. In a triangular membership function (5.11) b is a position parameter and a and c are both position and width parameters. Position parameters determine the relative position of the fuzzy sets in a fuzzy partition and width parameters determine the overlap of the fuzzy sets.

In order not to apply computationally expensive constraint satisfaction strategies we can use the following approach to meet the four most common constraints while applying the updates by Algorithm 5.4.

(i) *Valid parameters*: If updating the parameters would lead to invalid values the updates are corrected. For example, for a triangular membership function (5.11) $l \leq a \leq b \leq c \leq u$ must always hold, where $[l, u]$ is the domain of the corresponding variable.

(ii) *Do not pass*: If an update causes a position parameter of the current fuzzy set to become smaller than the corresponding parameter of the left neighbour or larger than the corresponding parameter of the right neighbour, the update is corrected.

(iii) *Must overlap*: If an update causes a width parameter of the current fuzzy set to change in such a way that the current fuzzy set no longer sufficiently overlap with one of its neighbours, the update is corrected.

(iv) *Stay symmetric*: If the parameter updates of the current fuzzy set lead to an undesired asymmetry, the update is corrected.

Algorithm 5.2: Compute updates for consequent fuzzy sets

ComputeConsequentUpdate (ν, e, τ, t)

(The algorithm obtains the following input parameters: *)*
(ν: fuzzy set for which parameter updates must be computed *)*
(e: error value *)*
(τ: degree of fulfilment of the rule that uses ν in the consequent *)*
(t: current target value from the domain of ν*)*
(a, b, c, d are parameters of ν, see (5.11) - (5.13)*)*

1: **if** (ν is triangular) **then** *(* **triangular fuzzy set** *)*
2: shift $= \sigma \cdot e \cdot (c - a) \cdot \tau \cdot (1 - \nu(t))$;
3: $\Delta b = \Delta b +$ shift;
4: **if** $\nu(t) > 0$ **then** *(* $t \in$ support(ν), focus ν on t *)*
5: $\Delta a = \Delta a + \sigma \cdot \tau \cdot (b - a) +$ shift;
6: $\Delta c = \Delta c - \sigma \cdot \tau \cdot (c - b) +$ shift;
7: **else** *(* $t \notin$ support(ν), shift ν to cover t *)*
8: $\Delta a = \Delta a + \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (b - a) +$ shift;
9: $\Delta c = \Delta c + \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (c - b) +$ shift;
10: **end if**
11: **else if** (ν is trapezoidal) **then** *(* **trapezoidal fuzzy set** *)*
12: shift $= \sigma \cdot e \cdot (d - a) \cdot \tau \cdot (1 - \nu(t))$;
13: **if** $\nu(t) > 0$ **then** *(* $t \in$ support(ν), focus ν on t *)*
14: $\Delta a = \Delta a + \sigma \cdot \tau \cdot (b - a) +$ shift;
15: $\Delta b = \Delta b + \sigma \cdot \tau \cdot (c - b) +$ shift;
16: $\Delta c = \Delta c - \sigma \cdot \tau \cdot (c - b) +$ shift;
17: $\Delta d = \Delta d - \sigma \cdot \tau \cdot (d - c) +$ shift;
18: **else** *(* $t \notin$ support(ν), shift ν to cover t *)*
19: $\Delta a = \Delta a + \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (b - a) +$ shift;
20: $\Delta b = \Delta b + \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (c - b) +$ shift;
21: $\Delta c = \Delta c + \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (c - b) +$ shift;
22: $\Delta d = \Delta d + \text{sgn}(t - b) \cdot \sigma \cdot \tau \cdot (d - c) +$ shift;
23: **end if**
24: **else if** (ν is bell-shaped) **then** *(* **bell-shaped fuzzy set** *)*
25: shift $= \sigma \cdot e \cdot a \cdot \tau \cdot (1 - \nu(t))$;
26: $\Delta b = \Delta b +$ shift;
27: **if** $\nu(t) > e^{-c^2}$ **then** *(* $t \in$ support(ν), focus ν on t *)*
28: $\Delta a = \Delta a -$ shift;
29: **else** *(* $t \notin$ support(ν), shift ν to cover t *)*
30: $\Delta a = \Delta a +$ shift;
31: **end if**
32: **end if**

Algorithm 5.3: Compute antecedent fuzzy set updates

ComputeAntecedentUpdates(μ, e, p)

(The algorithm obtains the following input parameters: *)*
(μ: fuzzy set for which parameter updates must be computed *)*
(e: error value *)*
(p: current input value from the domain of μ *)*
(a, b, c, d are parameters of μ, see (5.11) - (5.13)*)*

1: **if** $(e < 0)$ **then** *(* take degree of membership into account *)*
2: $f = \sigma \cdot \mu(p)$ *(* σ is a learning rate *)*
3: **else**
4: $f = \sigma(1 - \mu(p))$
5: **end if**
6: **if** (μ is triangular) **then** *(* **triangular fuzzy set** *)*
7: $\text{shift} = f \cdot e \cdot (c - a) \cdot \text{sgn}(p - b)$;
8: $\Delta a = \Delta a - f \cdot e \cdot (b - a) + \text{shift}$; *(* lower bound of support *)*
9: $\Delta c = \Delta c + f \cdot e \cdot (c - b) + \text{shift}$; *(* upper bound of support *)*
10: $\Delta b = \Delta b + \text{shift}$; *(* center *)*
11: **else if** (μ is trapezoidal) **then** *(* **trapezoidal fuzzy set** *)*
12: **if** $(b \leq p \leq c)$ **then** *(* $p \in core(\mu)$ *)*
13: $\Delta b = \Delta b - f \cdot e \cdot (c - b)$; *(* lower bound of core *)*
14: $\Delta c = \Delta c + f \cdot e \cdot (c - b)$; *(* upper bound of core *)*
15: **else** *(* $p \notin core(\mu)$ *)*
16: $\text{shift} = f \cdot e \cdot (d - a) \cdot \text{sgn}(p - b)$;
17: $\Delta a = \Delta a - f \cdot e \cdot (b - a) + \text{shift}$; *(* lower bound of support *)*
18: $\Delta b = \Delta b - f \cdot e \cdot (c - b) + \text{shift}$; *(* lower bound of core *)*
19: $\Delta c = \Delta c + f \cdot e \cdot (c - b) + \text{shift}$; *(* upper bound of core *)*
20: $\Delta d = \Delta d + f \cdot e \cdot (d - c) + \text{shift}$; *(* upper bound of support *)*
21: **end if**
22: **else if** (μ is bell-shaped) **then** *(* **bell-shaped fuzzy set** *)*
23: $\text{temp} = f \cdot e \cdot b$;
24: $\Delta a = \Delta a + \text{temp}$; *(* width of α-cut at e^{c^2} *)*;
25: $\Delta b = \Delta b + \text{temp} \cdot \text{sgn}(p - b)$; *(* center *)*
26: **end if**

Algorithm 5.4: Carry out the update of a fuzzy set

Update (μ, k)

(The algorithm obtains the following input parameters: *)*
(μ: fuzzy set for which parameter updates must be computed *)*
(k: $k = 1$ for online learning, $k = s$ for batch learning *)*

1: **if** (the update of μ conflicts with the constraints for μ) **then**
2: modify the updates of μ such that the constraints are satisfied;
3: **end if**

4: **for all** parameters w of μ **do**
5: $w = w + \Delta w/k$; *(* apply parameter updates *)*
6: $\Delta w = 0$; *(* reset parameter updates *)*
7: **end for**

Example 5.4 In Figure 5.8 an example for online learning is shown. We assume that the fuzzy set in the middle would obtain the form shown in part (1) of Figure 5.8, if the updates computed during the last training step were applied. We further assume that fuzzy sets must have valid parameters, must not exchange the positions with adjacent fuzzy sets, must overlap with adjacent fuzzy sets and must stay symmetric. Figure 5.8 illustrates the application of the four steps given in the list above.

If we used batch learning, we would obtain a different solution, because we would have to process the fuzzy sets either from left to right or from right to left in order to avoid an iterated constraint satisfaction procedure. In this case we would have to extend the support of the leftmost fuzzy set in order to ensure overlapping and then we would have to make it symmetric again. ◇

The most restrictive constraint for the training algorithm is the requirement that the degrees of membership add up to 1 for each value of a domain. In contrast to the above-mentioned constraints, this restriction can obviously not be enforced by simply correcting the parameter modifications of the currently updated fuzzy set. The left and right neighbours in the fuzzy partition must also be modified. This feature cannot be implemented by the update procedure in Algorithm 5.4. An additional subroutine must be defined for this and must be called after updating all fuzzy sets of a variable.

The easiest way to implement this constraint is to use online learning. In this case it is sufficient to correct the left and right neighbour of a fuzzy set after updating it. This approach was implemented, for example, in the software tools NEFCLASS-PC [Nauck et al., 1996, Nauck and Kruse, 1997b] and NEFCLASS-X [Nauck

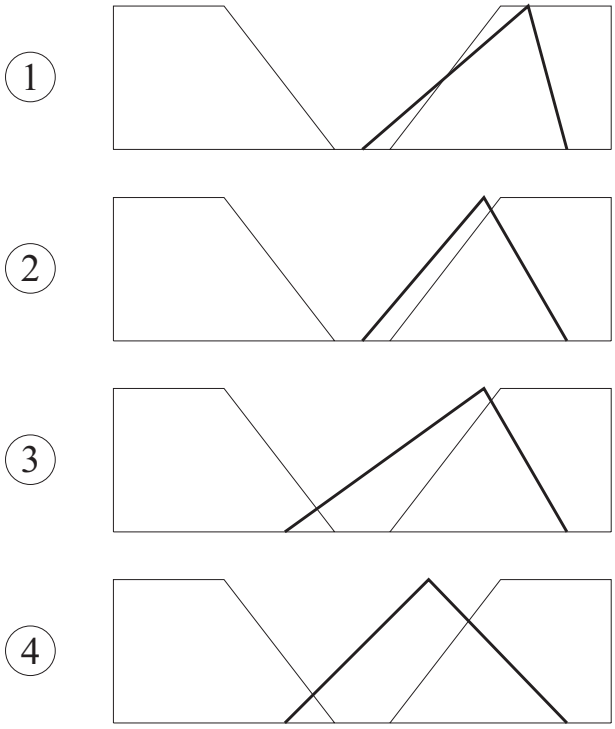


Figure 5.8: Constraint satisfaction after computing updates by online learning: (1) the second fuzzy set would not satisfy the constraints if the updates were applied, (2) restore relative positions, (3) restore overlap, (4) restore symmetry

and Kruse, 1998b].

If, however, batch learning is used, then we must correct the whole fuzzy partition after each epoch. To simplify the computation we process the fuzzy partition from left to right (or from right to left). Two consecutive fuzzy sets of the partition are considered and the overlapping between them is adjusted such that the membership degrees add up to 1. After each pair of consecutive fuzzy sets has been processed, the fuzzy partition satisfies the constraint.

The adjustment should take into account the shift of the core of the fuzzy sets, which was caused by the previous parameter updates. This means, if possible, the constraint is enforced by first trying to adjust width parameters; only if that fails are position parameters adjusted. This approach is implemented, for example, in the software tool NEFCLASS-J [Nauck et al., 1999].

Example 5.5 Figure 5.9 shows an example for correcting a fuzzy partition during batch learning such that the degrees of membership add up to 1. We assume that the fuzzy partition looks like part (1) of Figure 5.9 after Algorithm 5.4 has been executed. We further assume that the membership functions need not be symmetric.

To correct the partition we begin at the lower bound and consider two adjacent fuzzy sets. The arrows above the fuzzy sets in part (1) of Figure 5.9 denote the direction in which the cores of the fuzzy sets have moved due to the recent update step. In part (2) we adjust the overlapping between the first and second fuzzy set and in part (3) the overlapping between the second and third fuzzy set is corrected. We take the shift of the cores into account and try to maintain or – if that is not possible – to increase the shift while correcting the parameters of the membership functions. ◇

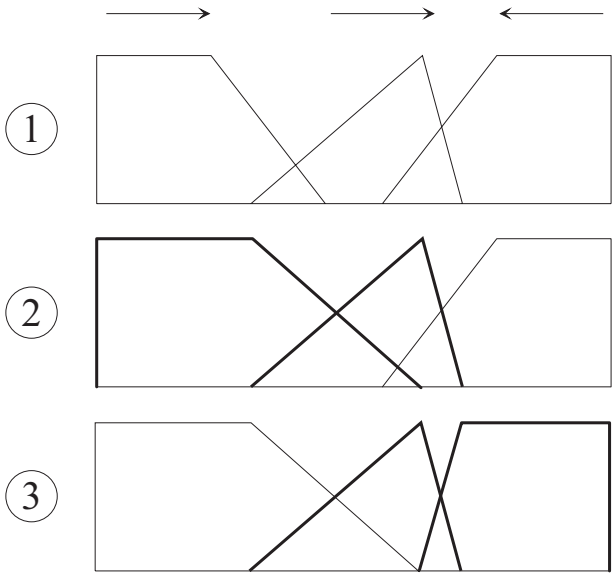


Figure 5.9: Correcting a partition after an update step in batch learning (1) by adjusting the overlap between each two adjacent fuzzy sets (2 and 3) in order to satisfy the constraint "add up to 1"

5.4 Fuzzy Classifiers

In this section we consider learning in fuzzy classifiers. The degree of fulfilment of a rule is interpreted as the degree of membership of the input pattern to the class that is referred to in the consequent (Definition 3.5).

This means learning in a fuzzy classifier only needs to tune antecedent parameters. Each fuzzy classification rule can be interpreted as a (labelled) fuzzy cluster in the input space. Learning then corresponds to changing the size and location of the clusters. The number and initial locations of the clusters have already been determined by a rule learning procedure (Chapter 4).

If a fuzzy classifier is implemented by using differentiable membership functions and a differentiable t-norm like the product, then gradient descent methods to determine the antecedent parameters can be used (Section 5.2). In this section, however, we do not make this assumption and view a fuzzy classifier as a simplified Mamdani-type fuzzy system (Definition 3.5). This means, we use the min operation to determine the degree of fulfilment of a rule, and use the max operation to determine the degree of membership of some pattern to a class. We also allow non-continuous forms of membership functions like triangles and trapezoids and fuzzy sets over symbolic variables.

This means, as in Mamdani-type fuzzy systems, we cannot use gradient descent procedure for training. Instead we will apply the same heuristics as in Section 5.3.

The learning algorithm 5.5 can be applied to any fuzzy classifier. The algorithm uses the same notations and options as Algorithm 5.1 (compare p. 98). In addition the following notations are used:

- $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$: a training pattern consists of an input pattern $\mathbf{p} \in X$ and a target vector $\mathbf{t} \in [0,1]^m$. In contrast to Algorithm 5.1 we do not require $X = \mathbb{R}^n$ and also allow symbolic variables $x_i \in X_i$, where X_i is a finite set. For numeric variables we have $x_j \in X_j \subseteq \mathbb{R}$. A target vector $\mathbf{t} \in [0,1]^m$ of a training pattern $(\mathbf{p}, \mathbf{t}) \in \tilde{\mathcal{L}}$ denotes a (fuzzy) classification of an input pattern $\mathbf{p} \in X$.
- c_j: output variable that stores the degree of membership of an input pattern to class j.
- $\mathrm{con}(R)$: index that specifies the class used in the consequent of rule R.

The end criterion that terminates Algorithm 5.5 can depend on any combination of the following conditions:

- the number of misclassified patterns has been sufficiently decreased,
- the overall output error is small enough,

- the number of misclassified patterns and/or the overall output error have reached a local minimum,
- a maximum number of iterations has been exceeded.

It is useful to observe these conditions on a separate validation set and not on the training set in order to avoid over-generalization.

We must note that the number of misclsassifications can increase, if the overall error decreases. The error value measures how crisp the classification is. Let us consider a simplified example of two patterns $\mathbf{p}_1$ and $\mathbf{p}_2$ with class($\mathbf{p}_1$) = c_1 and class($\mathbf{p}_2$) = c_2 as shown in the following table.

	c_1	c_2	c_1	c_2
$\mathbf{p}_1$	0.51	0.49	0.90	0.30
$\mathbf{p}_2$	0.49	0.51	0.51	0.49
SSE	0.96		0.62	
miscl.	0		1	

We can see that the sum of squared errors decreases, while the number of misclassifications increases. An end criterion for training the membership function can take both indicators – the error and the number of misclassifications – into account. A classifier with a slightly higher number of misclassifications but a substantially smaller error may be more desirable.

The modification of the antecedent fuzzy sets follows the same guidelines that are used in Section 5.3 for Mamdani-type fuzzy systems. We need to specify an error value for each individual rule that depends on the output error for the class used in the consequent and from the current degree of fulfilment τ_r (compare (5.10)). The classification of an input pattern is obtained by partitioning the input space with fuzzy clusters which correspond to the rules of the classifier. An optimal classifier would output 1 for the correct class and 0 for all other classes. This means we want the partitioning of the input space to be as crisp as possible. The rule error should be maximum for $\tau_r = 0.5$ in order to force a rule to produce degrees of fulfilment near 1 or 0.

In analogy to (5.10) we could use

$$\mathcal{E}_r = \tau_r(1 - \tau_r)\ E_{\mathrm{con}(R_r)}$$

to compute the rule error. As in (5.10) the rule error would be zero for $\tau_r = 0$ and $\tau_r = 1$. A rule with $\tau_r = 0$ that does not contribute to the current classification would not be trained. A rule with $\tau_r = 1$ perfectly matches the current input pattern. If such a rule causes a misclassification, we have no means to correct the output, because the consequent of a rule does not have any adaptable parameters. One possibility to overcome this problem is to use rule weights. However, as we

Algorithm 5.5: Fuzzy set learning in fuzzy classifiers

```
1: repeat
2: for all patterns (p, t) ∈ L̃ do                    (* there are s training patterns *)
3:    propagate the next training pattern (p, t);
4:    for each output variable c_j do
5:       compute the output error E_j;                 (* based on t_j − o_j, see page 96 *)
6:    end for

7:    for each rule R_r with A_r(p) > 0 do
8:       j = con(R_r);
9:       E_r = (A_r(p) · (1 − A_r(p)) + ε) · E_j;          (* rule error 5.14 *)

10:      if (MINFSONLY) then          (* modify only one fuzzy set in each rule *)
11:         k = argmin_{i∈{1,...,n}} {μ_r^(i)(p_i)};
12:         ComputeUpdates(μ_r^(k), E_r);                  (* see Algorithm 5.6 *)
13:         if (ONLINELEARNING) then          (* update after each pattern *)
14:            Update (μ_r^(k), 1);                        (* see Algorithm 5.4 *)
15:         end if
16:      else                                 (* modify all fuzzy sets in each rule *)
17:         for all μ_r^(i) do
18:            ComputeUpdates(μ_r^(i), E_r);               (* see Algorithm 5.6 *)
19:            if (ONLINELEARNING) then       (* update after each pattern *)
20:               Update (μ_r^(i), 1);                     (* see Algorithm 5.4 *)
21:            end if
22:         end for
23:      end if
24:   end for                                          (* end for each rule *)
25: end for(* all training patterns *)

26: if (BATCHLEARNING) then                  (* update after complete epoch *)
27:    for all input variables x_i do
28:       for all fuzzy sets μ_j^(i) do
29:          Update (μ_j^(i), s);                        (* see Algorithm 5.4 *)
30:       end for
31:    end for
32: end if
33: until end criterion;
```

pointed out in Section 5.1, we prefer not to use weighted rules due to the semantical problems they cause.

Another possibility is not to train such a rule and to assume that the current pattern is an outlier that should not influence the training procedure. However, this means, that ill-defined rules cannot be corrected by the training procedure. Therefore we compute the rule error by

$$\mathcal{E}_r = (\tau_r(1-\tau_r)+\varepsilon)\ E_{\mathrm{con}(R_r)}, \tag{5.14}$$

where ε is a small positive number, e.g. $\varepsilon = 0.01$. By doing this we can ensure, that the rule error is only zero, if the output error is zero. Rules with a degree of fulfilment of 0 or 1 are also trained to a small extent and thus we compensate for the absence of adaptable consequent parameters. This means that a fuzzy cluster that corresponds to a rule can be moved even it it is located in an area of the input space with no data, or if it exactly matches certain outliers.

The procedure *ComputeUpdates* (Algorithm 5.6) must provide the necessary computations for triangular (5.11), trapezoidal (5.12) and bell-shaped membership functions (5.13). Because computations are identical to the computations in Mamdani-type fuzzy systems, we can call *ComputeAntecedentUpdates* (Algorithm 5.3, Section 5.3) in this case. In addition we consider fuzzy sets of the form

$$\begin{aligned} &\mu : X \to [0,1], \quad \mu = \{(x_1, a_1), \ldots, (x_q, a_q)\}, \text{with} \\ &\mu(x_i) = a_i, \quad a_i \in [0,1], \quad x_i \in X, \quad i \in \{1, q\} \end{aligned} \tag{5.15}$$

where X is some finite domain. Such kinds of membership functions are used to represent fuzzy sets for symbolic variables. We call this kind of representation a *list fuzzy set*. The necessary computations to update such fuzzy sets are given in Algorithm 5.6. To apply the computed updates, we can use Algorithm (5.4) from Section 5.3.

Algorithm 5.6: Compute fuzzy set updates for a fuzzy classifier

ComputeUpdates(μ, e, p)

(The algorithm obtains the following input parameters: *)*
(μ: fuzzy set for which parameter updates must be computed *)*
(e: error value *)*
(p: current input value from the domain of μ *)*

```
1: if (μ is triangular or trapezoidal or bell-shaped) then
2:    ComputeAntecedentUpdate(μ, e, p);          (* see Algorithm 5.3 *)
3: else if μ is a list then                      (* list fuzzy set, see (5.15) *)
4:    if (e < 0) then                  (* take degree of membership into account *)
5:       f = σ · μ(p)                                  (* σ is a learning rate *)
6:    else
7:       f = σ(1 − μ(p))
8:    end if
9:    Δμ[p] = Δμ[p] + f · e;        (* modification for degree of membership of p *)
10: end if
```

5.5 Pruning Fuzzy Rule Bases

In order to improve the readability of a fuzzy rule base derived by a learning process pruning techniques can be used. Pruning techniques are well-known from neural networks [Haykin, 1994, Neuneier and Zimmermann, 1998] and decision tree learning [Quinlan, 1993]. They are used to reduce the complexity of a model.

In neural networks, tests are made for the parameters of a network (weights or nodes) to determine how the error would change, if the parameter is removed. Frequently applied pruning strategies are, for example, OBD (optimal brain damage) [Le Cun et al., 1990] or EBD (early brain damage) [Neuneier and Zimmermann, 1998], which try to remove weights from a neural network. There are also pruning techniques known from decision tree learning [Quinlan, 1993].

Fuzzy rule base pruning can be based on a simple greedy algorithm that does not need to compute complex test values as in neural network pruning methods like OBD or EBD.

In order to prune a rule base we consider four heuristic strategies that can work in an automatic fashion without the necessity of user interaction.

(i) Pruning by correlation: The variable that has the smallest influence on the output is deleted. To identify this variable statistical measures like correlations and χ^2 tests or measures from information theory like the information gain can be used.

(ii) Pruning by classification frequency: The rule that yields the largest degree of fulfilment in the least number of cases is deleted. Such a rule is only responsible for the classification of a small number of patterns. If these patterns are also covered by other rules, the performance of the fuzzy rule base may not decrease. But if these patterns represent exceptions it may not be possible to delete the selected rule without a decrease in performance.

(iii) Pruning by redundancy: The linguistic term that yields the minimal degree of membership in an active rule in the least number of cases is deleted. This pruning strategy assumes that the min operator is used in order to evaluate the antecedent of a rule. In this case a term that always provides large degrees of membership, does not influence the computation of the degree of fulfilment and the term assumes the role of a don't care variable. This pruning strategy can also be applied, if other t-norms are used, e.g. the product, but it may be less effective in these cases.

(iv) Pruning by fuzziness: The fuzzy set with the largest support is identified and all terms that use this fuzzy set are removed from the antecedents of all rules. This pruning strategy is comparable to (iii), because it assumes that fuzzy sets with large supports provide large degrees of membership for many input values

and thus terms that use this fuzzy set do not influence the computation of the degree of fulfilment in a rule. Another justification for this strategy is that fuzzy sets actually get very large supports during training, if the corresponding variable has a large variance and is thus less useful for prediction.

An automatic pruning algorithm can be obtained by applying the four strategies consecutively. After each pruning step the membership functions should be trained again before a pruning step is declared a failure or a success. If a pruning step has failed, the rule base is restored to its previous state. This means the modifications caused by a pruning step are only kept, if the step has successfully improved the rule base. In the case of a classification problem, the pruning algorithm must take care not to remove the last rule for a class.

After an application step of one of the strategies has failed, it must be decided whether to carry on with this strategy and the next parameter it recommends for pruning, or to switch to the next pruning strategy. In order to reduce runtime, usually each of the four pruning strategies is iterated until a pruning step fails to improve the performance of the rule base. Then the next pruning strategy is selected. An implementation of this approach produces good results for neuro-fuzzy classification systems [Nauck et al., 1999].

The improvement of the rule base can be defined in terms of performance (i.e. reduction of error) and in terms of complexity or simplicity (i.e. number of parameters). There is usually a trade-off between performance and simplicity. To obtain high accuracy, a large number of free parameters is needed, which again result in a very complex and thus less comprehensible model. However, often the performance of a model can actually increase with the reduction of the number of parameters because the generalization capabilities of the model may increase. If the model has too many parameters, it tends to overfit the training data and displays poor generalization on test data. But if the number of parameters is too small, sufficient accuracy can no longer be attained.

The pruning algorithm can take this trade-off into account and can continue pruning, even if the performance decreases slightly, because a more comprehensible model is obtained. For implementation, a measure based on the minimum description length principle (MDL) [Rissanen, 1983] can be used. The MDL turned out to be well suited for decision trees [Kononenko, 1995] and favours models with few parameters and good performance. An approach where a pruning process for a rule base of a fuzzy classifier is guided by a measure based on the MDL can be found in [Klose et al., 1998].

If variables are deleted from the rule, the rule base can become inconsistent during pruning. This may happen for the above-mentioned pruning strategies (i), (iii) and (iv). Inconsistencies must be resolved by deleting some rules. If the rule learning algorithms from Section 4.2 are used, the performance values of the rules can be

used to select rules for deletion until the rule base is consistent again.

A consistent rule base is a rule base that does not contain contradictions or redundancies.

- A contradiction occurs if there are two rules with different consequents and if their antecedents are either equal or one antecedent is more general than the other. An antecedent A is more general than an antecedent B, if A contains fewer linguistic terms than B and all linguistic terms of A also appear in B.
- The rule base is redundant, if there are two rules with identical consequents and if the antecedent of one rule is more general than the antecedent of the other rule.

The rule base can be made consistent by identifying pairs of contradictory and/or redundant rules and deleting rules with smaller performance values.

5.6 Analysis of the Learning Algorithms

In this section we analyze the time and memory complexity of the fuzzy set learning algorithms 5.1 and 5.5 presented in Sections 5.3 and 5.4. We make the same assumptions and use the same notations as in Section 4.5.

We consider the following operations:

- computation of a degree of membership,
- computation of the parameter modifications of a fuzzy set,
- modification of a fuzzy set,
- comparison of degrees of membership.

The operations may have different costs, but the costs do not depend on the problem size which is given by the number of patterns s, the number of rules r, the number of input variables n, the number of output variables v and the number of free parameters per variable, which is estimated by q (Eq. 4.7).

The propagation of one pattern requires rn computations of degrees of membership and rv aggregations of the conclusions (by a maximum operation). For a function approximation problem we have to do a defuzzification. This is usually done by sampling the output domains and computing weighted sums. These costs only increase with the number of output variables and do not otherwise depend on the size of the problem. This means the number of operations for propagating a pattern is $rn + rv + v$.

The computation of the output error requires v operations and the cost for computing the updates for the consequent fuzzy sets are rv. The rule errors can be computed by rv operations, because the degrees of fulfilment for each rule are already determined by the propagation of the current pattern. To compute the updates for the antecedent fuzzy sets rn operations are required, because we either have to find the variable with the smallest degree of membership (MINFSONLY) or to process all variables.

If online learning is selected, we can regard the computation of the update values and their (constrained) application as one operation. In the case of batch learning we must add $2q(n + v)$ operations after all patterns are processed to consider the application of the parameter updates and the possible constraint that degrees of membership must add up to one for each value of a domain.

Altogether, in the case of batch learning, we obtain for the number of operations

$$f_t(s, r, n, v, q) \leq s(rn + rv + v + v + rv + rn) + 2q(n + v). \tag{5.16}$$

If we consider that only those membership functions must be updated which really occur in the fuzzy rules, we can simplify (5.16) and obtain

$$f_t(s, r, n, v) \leq sr(2n + 2v) + 2v + 2r(n + v) = O(sr(n + v)). \tag{5.17}$$

For a fuzzy classifier we set $v = 1$ and ignore all constant parts in the sum. In addition, we do not need to update output fuzzy sets. We obtain

$$f_t(s, r, n) \leq 2srn + 2rn = O(srn). \tag{5.18}$$

From these considerations we can conclude that training the membership functions of a fuzzy system has the complexity $O(srn)$, if we assume that n denotes the number of all variables with adaptive membership functions.

The predominant factor is the number of training patterns s. We can expect $r \ll s$ because we are interested in fuzzy systems with generalization capabilities and we can also expect $n \ll s$, because we demand $nq \leq s$ (see Section 4.5).

The memory requirements of algorithms 5.1 and 5.5 are the same as for the algorithms 4.1 and 4.4 analyzed in Section 4.5. In addition to the fuzzy set parameters we must also store their modifications, but this does not change the memory complexity and we obtain

$$f_m(s, n, q) = O(sn + sq). \tag{5.19}$$

The convergence of the training algorithms can only be guaranteed by a suitable end criterion for their outermost loop. We stop training after the error or the number of misclassifications have reached a local maximum from which the algorithm could not escape or if the number of cycles exceeds a certain limit. After training we

restore the best solution, we have obtained so far. By this we guarantee that the performance of the resulting fuzzy system is not worse than the performance of the initial fuzzy system.

The computation of the parameter updates is based on simple heuristics. The success of the training procedure depends on the initial fuzzy partitions, the rule base, the distribution of training patterns and the learning rate. For example, if we select a learning rate that is too large we can observe oscillations in the error just as in the training of neural networks. In Figure 5.10 we can see a training process of NEFCLASS-J using the Iris data set (see Chapter 6), where the learning rate was set to $\sigma = 10$. A value of $\sigma < 1$ would be appropriate for this learning problem.

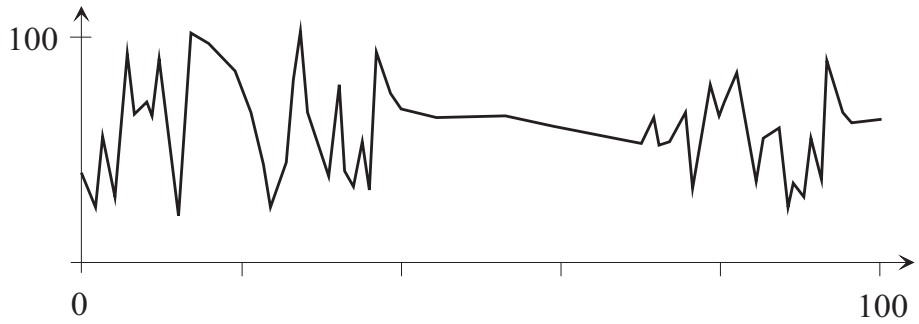


Figure 5.10: Oscillation of the error, if the learning rate is too large

Chapter 6

Data Analysis with NEFCLASS

In this chapter we show how a neuro-fuzzy system for a special application area – fuzzy classification – can be derived using the definition of a fuzzy perceptron from Chapter 3 and the learning algorithms provided in Chapters 4 and 5.

NEFCLASS (neuro-fuzzy classification) is a neuro-fuzzy approach to derive fuzzy classification rules from a set of labelled data [Nauck and Kruse, 1997b, Nauck and Kruse, 1998b, Nauck and Kruse, 1999b]. NEFCLASS creates a fuzzy classifier according to Definition 3.5 by using the structure-oriented rule learning algorithms 4.1 and 4.5 discussed in Sections 4.2 and 4.3. After the rule base was created, NEFCLASS trains the membership functions using Algorithm 5.5 given in Section 5.4. Subsequently the rule base is reduced by applying the pruning strategies described in Section 5.5.

Because NEFCLASS uses structure-oriented rule learning, each created fuzzy rule is a multi-dimensional fuzzy set whose support is a hyperbox in the data space. The degree of membership of a pattern in the multi-dimensional fuzzy set is the degree to which that pattern belongs to the class that was assigned to the multi-dimensional fuzzy set by the learning process. Adjacent rules overlap, i.e. a pattern can belong to several classes with non-zero degree of membership. The main goal of NEFCLASS is to create an interpretable fuzzy rule base. Therefore it uses constraints during fuzzy set learning as described in Section 5.3.

In the following section we show how NEFCLASS can be represented in a connection-oriented feed-forward network Afterwards we present some considerations concerning implementations. The last four sections discuss the applicaton of NEFCLASS to classification problems. In Section 6.3 we illustrate the effect of rule weights, which was discussed in detail in Section 5.1. In Sections 6.4 and 6.5 we use a medical data set with a few missing values to show how very compact and highly interpretable classifiers can be obtained by NEFCLASS. This chapter concludes with an example where NEFCLASS was applied as a preprocessing tool in image analysis. In this example the interpretability of the result is not important, but it shows, that neuro-fuzzy methods are applicable in complex real world problems.

6.1 Network Representation of NEFCLASS

A NEFCLASS system (Figure 6.1) can be represented as a special kind of fuzzy perceptron, because Definition 3.12 is flexible enough to allow for the representation of fuzzy classifiers.

Definition 6.1 *A NEFCLASS system represents a fuzzy classifier $F_{\mathcal{R}}$ according to Definition 3.5 with a set of class labels $C = \{c_1, \ldots, c_m\}$, $\top_1 = \min$, $\top_2 = \min$ and $\perp = \max$. A connection-oriented network representation of a NEFCLASS system is a fuzzy perceptron according to Definition 3.12 with the following specifications:*

(i) W, the network structure, is a partial mapping from $U \times U \to \mathcal{F}(\mathbb{R})$ and is given by

$$W(u,v) = \begin{cases} \mu_j^{(i)} & \text{if } u = x_i \ (i \in \{1,\ldots,n\}) \wedge v = R_j \ (j \in \{1,\ldots,r\}) \\ \mathbb{1}_{\{1\}} & \text{if } u = R_j \ \wedge v = c_{i_j} = \mathrm{con}(R_j) \\ & (j \in \{1,\ldots,r\}) \ (i_j \in \{1,\ldots,m\}) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

In addition every two connections with weights $W(u,v)$ and $W(u',v')$ $(u = u',\ v \neq v',\ u,u' \in U_1,\ v,v' \in U_2)$ become coupled connections, if $W(u,v) = W(u',v')$ holds.

(ii) The network input for the third layer is computed as follows:

$$\mathrm{NET}_u \ : \ ([0,1] \times \mathcal{F}(\mathbb{R}))^{U_2} \to \mathcal{F}(\mathbb{R}),$$

$$\mathrm{net}_u \ : \ \mathbb{R} \to [0,1], \mathrm{net}_u(y) = \max_{u' \in U_2} \{\min\{o_{u'}, W(u',u)(y)\}\}$$

for $u \in U_3$.

(iii) The output of a unit in the third layer is given by

$$O_u : \mathcal{F}(\mathbb{R}) \to \mathbb{R}, \ o_u = O_u(a_u) = \mathrm{defuzz}(a_u) = \mathit{height}(a_u)$$

for $u \in U_3$.

In Figure 6.1 a NEFCLASS system with two inputs, five rules and two classes is shown. The main difference between a fuzzy perceptron and a NEFCLASS system is that only one connection protrudes from each unit of the second layer to one unit of the third layer. This connection represents the connection between a rule unit and the class used in the consequent of the corresponding rule. As we have shown in Figure 3.3(b) these connections have a constant weight of 1, which actually means

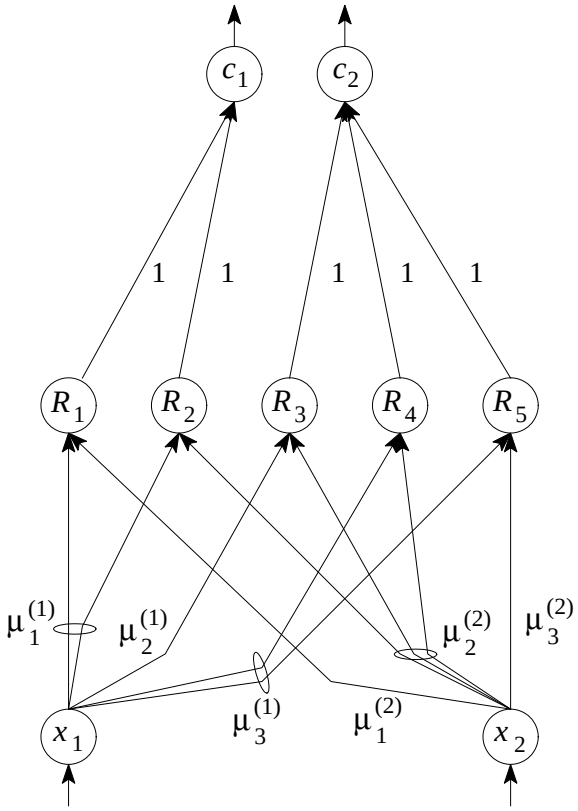


Figure 6.1: A connection-oriented network representation of a NEFCLASS system

they are not weighted. In order to keep the structure of a fuzzy perceptron that demands a fuzzy set attached to these connections we use the membership function $\mathbb{1}_{\{1\}}$ – which is the characteristic function of the set $\{1\}$ – to represent the singleton 1 as a fuzzy set (Definition 6.1(i)).

An output unit receives a modified version of this fuzzy set, i.e. its height is reduced to the maximum of the output values of all rule units connected to the considered output unit (Definition 6.1(ii)). The output values of the rule units are the degrees of fulfilment of the corresponding fuzzy rules. The output unit then defuzzifies this output fuzzy set by means of a special defuzzification function that computes the height of the output fuzzy set (Definition 6.1(iii)).

Because NEFCLASS uses coupled connections (shared weights), for each linguistic value there is only one representation as a fuzzy set. This ensures the interpretability of the fuzzy rule base. During learning it cannot happen that two fuzzy sets corresponding to the same label (e.g. *positive big*) develop differently. In Figure 6.1 shared weights are denoted by ellipses around the connections. Connections that share a weight always come from the same input unit.

6.2 Implementational Aspects

To be useful as a data analysis tool for classification problems, an implementation of NEFCLASS should provide the following features:

- fast generation of fuzzy classifiers through simple learning strategies,
- constrained fuzzy set learning to retain the interpretability of a generated classifier,
- automatic pruning to reduce the complexity of a generated classifier,
- automatic cross-validation to generate error estimates for a generated classifier,
- methods for the integration of prior knowledge and for the modification of a generated classifier by a user.

An overview over different implementations of NEFCLASS and their capabilities is given below.

Neuro-fuzzy approaches are a way to heuristically find parameters of fuzzy models by processing training data with a learning algorithm. Neuro-fuzzy approaches must be seen as development tools that can help to construct a fuzzy model. They are not "automatic fuzzy model creators". An implementation of a neuro-fuzzy approach must therefore enable a user to always supervise and interpret the learning process. We also have to keep in mind that as in neural networks the success for the learning process of a neuro-fuzzy model is not guaranteed. In addition the solution that is being built by a neuro-fuzzy tool is not only judged on its performance but also – if not especially – on its interpretability and simplicity. Thus the user must be able to build models fast by trying different parameter settings. This approach can be supported by using simple and fast learning procedures as discussed in Sections 4.2 and 5.4. Table 6.1 gives an overview on the algorithms involved in an implementation of NEFCLASS.

NEFCLASS is available in implementations for MS-DOS based Personal Computers (NEFCLASS-PC) [Nauck et al., 1996, Nauck and Kruse, 1997b] for Unix workstations under X-Window (NEFCLASS-X) [Nauck and Kruse, 1998b] and as a platform-independent Java application (NEFCLASS-J) [Nauck et al., 1999] that can run under any operating system for which a Java Virtual Machine (JVM) is available. Table 6.2 gives an overview on the features of the different implementations.

One of the most important aspects for implementing a neuro-fuzzy approach for generating interpretable fuzzy systems is that each linguistic value is represented by exactly one fuzzy set. This can be done, for example, by using an object-oriented programming language and specifying objects for fuzzy sets, fuzzy partitions and

Table 6.1: Learning algorithms involved in a NEFCLASS implementation

	remark	algorithms
rule learning	structure-oriented	
- in general		4.1, 4.2, 4.3
- for symbolic attributes	based on 4.1	4.5
- for missing values	based on 4.1	4.6
fuzzy set learning	heuristic approach	
- in general	based on 5.3	5.5, 5.6
- constraints	see p. 104	5.4
pruning	see Section 5.5	–

Table 6.2: Features of the different implementations of NEFCLASS

	NEFCLASS-PC	NEFCLASS-X	NEFCLASS-J
rule learning	yes	yes	yes
fuzzy set learning	yes	yes	yes
constraints for fuzzy sets	yes	yes	yes
can use prior knowledge	yes	yes	yes
online learning	yes	yes	yes
batch learning	no	no	yes
manual rule pruning	no	yes	no
automatic rule pruning	no	no	yes
symbolic attributes	no	no	yes
treatment of missing values	no	no	yes
cross validation	no	no	yes
programming language	Pascal	C++, TCL/TK	Java
operating system	MS-DOS	Unix (with X-Window)	any OS with JVM

fuzzy rules (Figure 6.2). Each object has fields to store data and methods to process data.

A list of fuzzy partition objects is created, one object per variable. Each fuzzy partition object contains a list of q_i fuzzy set objects. A fuzzy rule object contains a list of references to fuzzy sets to construct its antecedent (and consequent in the case of a function approximation system). By using references, we ensure that a linguistic expression that appears in several rules always uses the same fuzzy set. The fuzzy rule base is implemented as a list of fuzzy rule objects. In Figure 6.3 this approach is visualised. Note that rules R_1 and R_2 both use the linguistic expression "x_2 is $\mu_1^{(2)}$" and that therefore fuzzy set $\mu_1^{(2)}$ is shared by both rules. During learning

Object **Fuzzy Set**		Object **Fuzzy Partition**		Object **Fuzzy Rule**	
fields:	- parameters	fields:	- list of fuzzy sets	fields:	- references to fuzzy sets
	- updates		- ...		- error
	- membership				- performance
	- ...	methods:	- membership		- ...
methods:	- membership		- updates	methods:	- fulfilment
	- updates		-constraints		- error
	- contraints		- draw		- updates
	- draw		- ...		- performance
	- ...				- ...

Figure 6.2: Three central objects in a NEFCLASS implementation

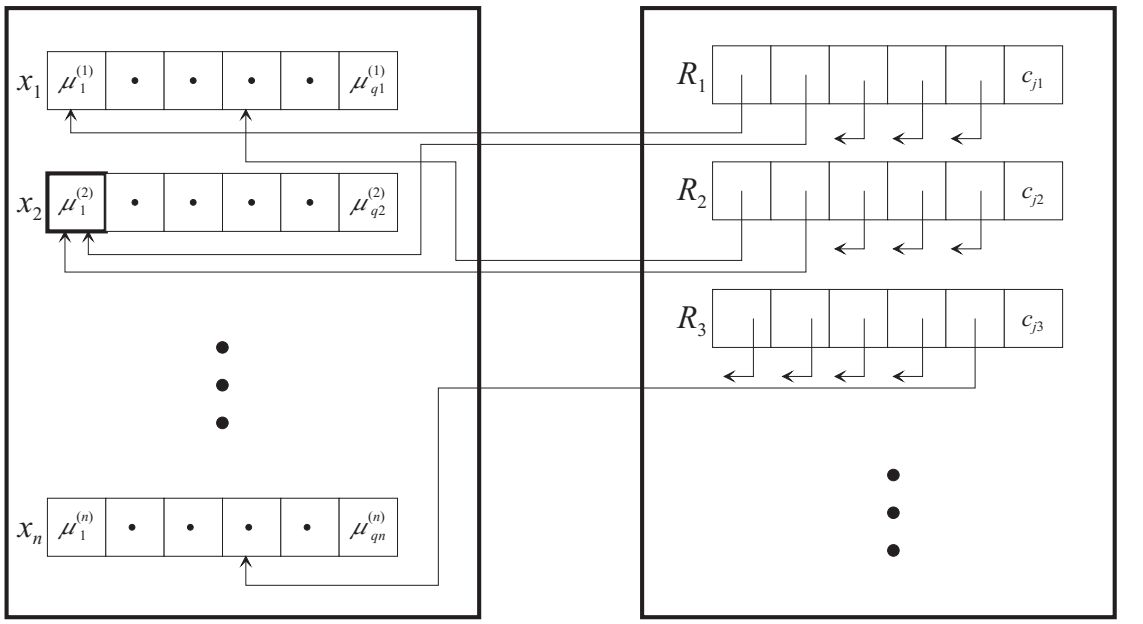


Figure 6.3: Implementation of a fuzzy rule base in NEFCLASS – note that fuzzy set $\mu_1^{(2)}$ is shared by rules R_1 and R_2

both rules R_1 and R_2 generate training signals that are used to modify $\mu_1^{(2)}$. Both rules always use the same version of $\mu_1^{(2)}$ for computing degrees of fulfilment.

We will demonstrate some of the learning capabilities of neuro-fuzzy systems in the following four sections using the most recent implementation NEFCLASS-J, which provides the most features.

Table 6.3: The rule base of the two experiments

	sepal length	sepal width	petal length	petal width	class
R_1:	*small*	*medium*	*small*	*small*	Iris Setosa
R_2:	*medium*	*small*	*medium*	*medium*	Iris Versicolor
R_3:	*medium*	*small*	*large*	*large*	Iris Virginica
R_4:	*large*	*medium*	*large*	*large*	Iris Virginica
R_5:	*large*	*small*	*large*	*large*	Iris Virginica

6.3 Effects of Rule Weights

To illustrate the effect of rule weights we use a simple example. We apply the software tool NEFCLASS-J [Nauck et al., 1999] to the well-known Iris data set[1] [Fisher, 1936], which consists of 150 patterns with four features. There are three classes with 50 patterns each. One class is linearly separable from the other two classes and two classes slightly overlap. The Iris data can be very easily classified by almost any classification method and is very often used as a benchmark. Therefore it is suitable for demonstrating the effect of rule weights. The learning outcome that is presented below is not the best that can be obtained with NEFCLASS for this data set. But we need a learning outcome where we can compare weighted and unweighted rules and so the result presented is adequate. The best result that can be obtained for the Iris data is briefly mentioned at the end of this section.

For the following two experiments three fuzzy sets (*small, medium* and *large*) for each variable are specified. The membership functions for *small* (*large*) are equidistant triangles (Eq. 5.11) and the leftmost and rightmost triangles are "shouldered". This means that for the fuzzy set *small* $\mu_{\text{small}}(a) = 1$ and for the fuzzy set *large* $\mu_{\text{large}}(c) = 1$ holds. The membership function for *medium* is a regular triangle. The degrees of membership for any value of the domain add up to 1 in the beginning. In both experiments the best five rules (see Algorithm 4.2) were selected from the rule learning procedure.

In the first experiment the rule base is trained without rule weights by just adapting the parameters of the membership functions. The second experiment additionally uses adaptive rule weights. In the second experiment the parameters of the membership functions must be also trained, because a sufficient learning outcome cannot be found if only rule weights are trained.

The rule base that is found by NEFCLASS-J is shown in Table 6.3. The learning process randomly selects 50% (stratified sample) of the complete data set for train-

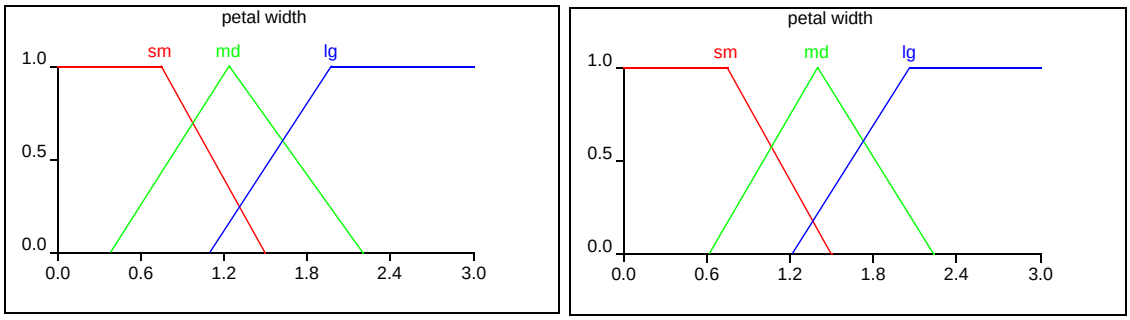
[1]The data set is available at the machine learning repository of the University of Irvine at ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

Table 6.4: Learning results for the two experiments on the Iris data set

errors			rule weights				
train.	test	epochs	R_1	R_2	R_3	R_4	R_5
3	3	200	–	–	–	–	–
4	2	70	1.58	1.59	1.24	1.13	1.23

ing. The other half of the data set is used for testing. The remaining parameter settings of the software are: learning rate = 0.01, online learning, maximum number of epochs = 200, stop training 30 epochs after a local minimum of the error was reached, and in each rule only the fuzzy set that provides the smallest degree of membership is trained.

Figure 6.4 shows the fuzzy sets of the variable "petal width" after training. In Figure 6.4(a) the fuzzy sets of the first experiment (unweighted rules) are shown. Figure 6.4(b) shows the fuzzy sets of the second experiment (weighted rules), where the rule weights are still attached to the rules and are not yet replaced by modifying the membership functions. Replacing the weights is considered below. As we can see the fuzzy sets are similar in both cases. However, the learning results are slightly different as shown in Table 6.4.



(a) (b)

Figure 6.4: Fuzzy sets for variable "petal width" after learning without rule weights (a) and with rule weights (b); in (b) the rule weights are not yet replaced

We do not want to discuss the quality of the learning outcome. In order to demonstrate the effect of rule weights it is only important that the results are similar. We consider rules R_3 and R_4 after the second experiment, where we used adaptive rule weights. Both rules use "petal width is *large*" in the antecedent.

As discussed in Section 5.1, we can now replace the weighted rules by equivalent unweighted rules with modified membership functions in the antecedents. For the

rules of the second experiment we obtain for R_3

$$large : \mathbb{R} \longrightarrow [0, 1.24],$$

and for R_4 we have

$$large : \mathbb{R} \longrightarrow [0, 1.13].$$

This means we now have two different interpretations for "petal width is *large*" in the result of the second experiment. There is only one representation of *large* in the first experiment, where no rule weights are used. Strictly speaking, in the second result *large* is no longer a fuzzy set. Even if we normalize the weights – which we can do in this case without changing the performance of the classifier – we still have the problem of two different non-normal fuzzy sets for the same linguistic term.

As the result for the first experiment shows, we can obtain an acceptable learning outcome without using rule weights and enjoy the benefit of an interpretable solution.

The best result (without rule weights) that can be obtained for the Iris data set by NEFCLASS-J uses three rules and only the variable *petal width*:
if *petal width* is *small* then class is *Iris Setosa*
if *petal width* is *medium* then class is *Iris Versicolor*
if *petal width* is *large* then class is *Iris Virginica*
This rule base causes 6 errors on the whole data set of 150 patterns. It was also discovered in 6 validation cycles of a 10-fold cross validation. The 99% confidence interval (Eq. 5.7) for the estimated error for unseen data that is computed by cross validation is $2.7\% \pm 2.8\%$. The membership functions for *petal width* are shown in Figure 6.5

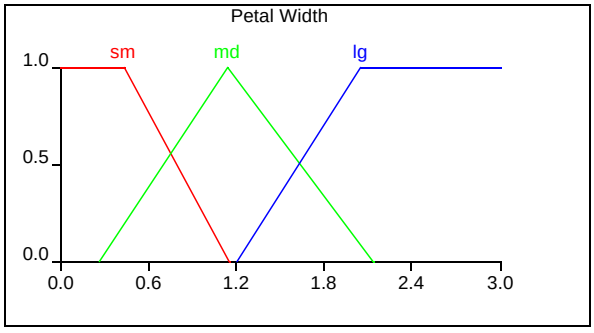


Figure 6.5: Membership function of variable *petal width* for the best learning result obtained by NEFCLASS-J for the Iris data (no rule weights are used)

6.4 Creating Small Classifiers

In this section we consider the creation of small well interpretable classifiers. We use the "Wisconsin Breast Cancer" (WBC) data set[2] to illustrate rule learning and automatic pruning of a rule base. The WBC data set is a breast cancer database that was provided by W.H. Wolberg from the University of Wisconsin Hospitals, Madison [Wolberg and Mangasarian, 1990]. The data set contains 699 cases and 16 of these cases have missing values. Each case is represented by an id number and 9 attributes (x_1: clump thickness, x_2: uniformity of cell size, x_3: uniformity of cell shape, x_4: marginal adhesion, x_5: single epithelial cell size, x_6: bare nuclei, x_7: bland chromatin, x_8: normal nucleoli, x_9: mitoses). All attributes are from the domain $\{1, \ldots, 10\}$. Each case belongs to one of two classes (benign: 458 cases, or malignant: 241 cases).

The goal of this experiment is to obtain a very small classifier. Therefore we use two fuzzy sets (*small* and *large*) per variable. The membership functions are shouldered triangles (see p. 127 and Figure 6.6).

Training is done with 10-fold cross validation. During rule learning the best two rules per class are selected and the membership functions are trained until the error on the validation set reaches a local minimum which the algorithm cannot escape within 30 epochs. The maximum number of training cycles is set to 200 epochs. After training the classifier is automatically pruned (compare Section 5.5).

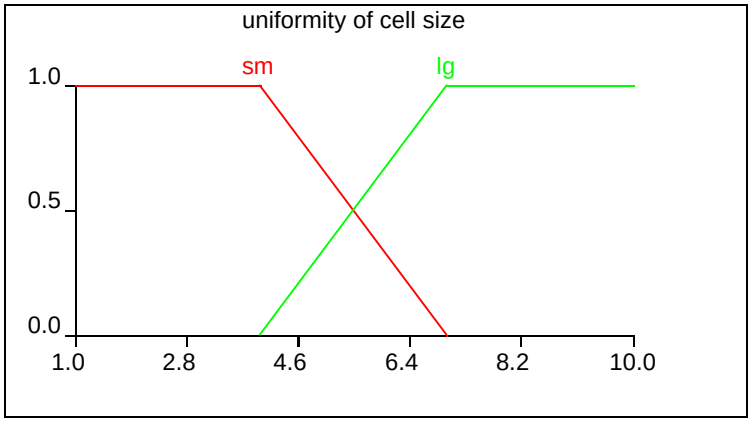


Figure 6.6: Initial membership functions for the variables of the WBC data set

With the described settings we create a fuzzy partition of $2^9 = 512$ overlapping hyperboxes on the data space. This means there are 512 possible fuzzy rules. During the 11 runs of the training process (10 validation runs and a final run to create

[2]The data set is available at the machine learning repository of the University of Irvine at ftp://ftp.ics.uci.edu/pub/machine-learning-databases.

the classifier) NEFCLASS-J detects between 127 and 137 rules that actually cover training data. From these rule base candidates the best two rules per class are selected in each run. This means that before pruning a classifier consists of four rules using nine variables.

The classifier that is created based on the complete data set contains the following two rules:

if uniformity of cell size (x_2) is *large* and
uniformity of cell shape (x_3) is *large* and
bare nuclei is (x_6) *large*
then class is *malignant*

if uniformity of cell size (x_2) is *small* and
uniformity of cell shape (x_3) is *small* and
bare nuclei is (x_6) *small*
then class is *benign*

During cross validation five different rule bases were created, each consisting of two rules. Four rule bases just used one variable (x_3 or x_6). The variables x_3 and x_6 were present in eight and nine rule bases respectively. x_2 appeared only once in a rule base during cross validation. The training protocol reveals, that it was not pruned from the final rule base, because the error slightly increases and one more misclassification occurs if x_2 is removed.

The mean error that was computed during cross validation is 5.86% (minimum: 2.86%, maximum: 11.43%, standard deviation: 2.95%). The 99% confidence interval (Eq. 5.7) for the estimated error is computed to $5.86\% \pm 2.54\%$. This provides an estimation for the error on unseen data processed by the final classifier created from the whole data set.

On the training set with all 699 cases the final classifier rules causes 40 misclassifications (5.72%), i.e. 94.28% of the patterns are classified correctly. There are 28 errors and 12 unclassified patterns which are not covered by one of the two rules. The confusion matrix for this result is given in Table 6.5. If one more misclassification can be tolerated, we can also delete variable x_2 from both rules. In this case 41 patterns are misclassified (32 errors and 9 unclassified patterns).

The linguistic terms *small* and *large* for each variable are represented by membership functions that can be well associated with the terms, even though they intersect at a slightly higher membership degree than 0.5 (Figure 6.7)

Table 6.5: The confusion matrix of the final classifier obtained by NEFCLASS-J

	Predicted Class							
	malignant		benign		not classified		sum	
malignant	215	(30.76%)	15	(2.15%)	11	(1.57%)	241	(34.48%)
benign	13	(1.86%)	444	(63.52%)	1	(0.14%)	458	(65.52%)
sum	228	(32.62%)	459	(65.67%)	12	(1.72%)	699	(100.00%)

correct: 659 (94.28%), misclassified: 40 (5.72%), error: 70.77.

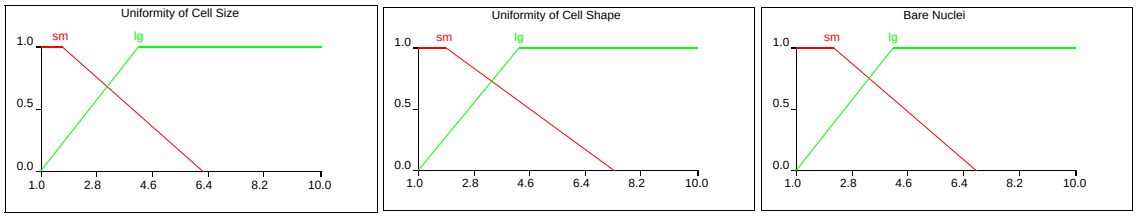


Figure 6.7: The membership functions for the three variables used by the final classifier

6.5 Using Symbolic Variables

This section illustrates the learning algorithm for mixed fuzzy rules (Algorithm 4.5), which is discussed in Section 4.3.

The WBC data from the previous section is also used for this experiment, because the values of all nine variables are actually from an ordinal scale. Classifiers usually simply treat them as metric values and good classification results can be obtained this way (see Table 6.7). To illustrate Algorithm 4.5 we chose to interpret variables x_3 and x_6 as categorical variables and the rest as metric variables. x_3 and x_6 are selected, because these two variables usually turn out to be influential in other classification approaches. They have been also included in almost all rule bases created during cross validation in the experiment from the previous section. All other parameters in NEFCLASS-J are set to the same values as in Section 6.4.

We use a 10-fold cross validation, and let the tool select the best two rules per class during rule learning. For each metric variable two initial membership functions are given (shouldered triangles, compare Figure 6.6). The fuzzy sets for the categorical variables are created during rule learning. The fuzzy sets are trained until the error on the validation set cannot be further decreased, but not longer than 200 cycles.

The final classifier contains only two rules using one and two variables, respectively (Figure 6.8):

(i) if x_2 (uniformity of cell size) is *small* and x_6 (bare nuclei) is $term_1^{(6)}$ then *benign*

(ii) if x_2 (uniformity of cell size) is *large* then *malignant*

The membership functions after training are shown in Figure 6.9. The fuzzy set for the categorical variable x_6 is drawn as a histogram. Its exact representation is

$$term_1^{(6)} = \{(1, 1.0), (2, 1.0), (3, 0.66), (4, 0.37), (5, 0.61), (6, 0.0), (7, 0.01), (8, 0.01), (9, 0.0), (10, 0.14)\}.$$

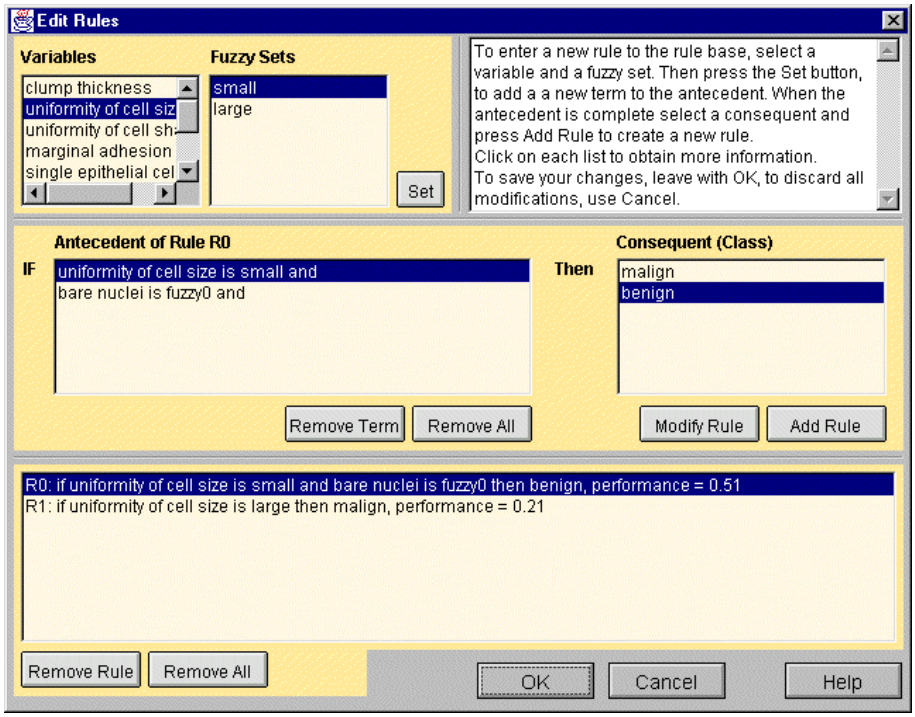
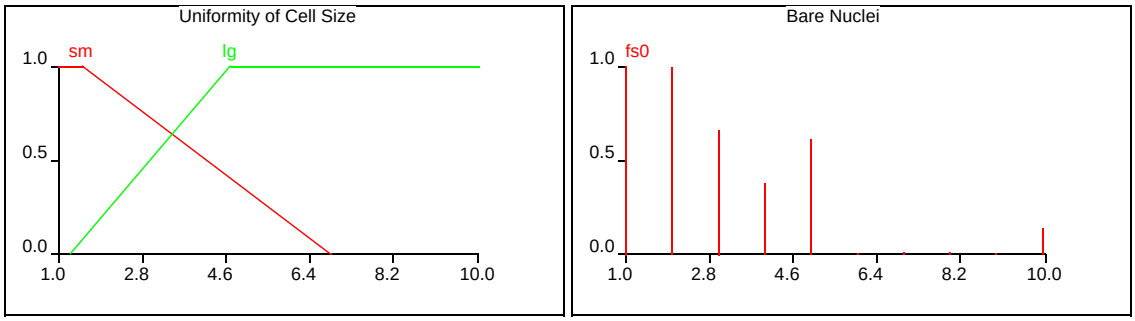


Figure 6.8: The rule editor of NEFCLASS-J displaying the learning result

This classifier causes 28 misclassifications (4.01%) on the *training data*, i.e. its classification rate is 95.99% (see Table 6.6). This classifier covers all data, there are no unclassified cases. The error estimation for *unseen data* obtained from cross validation yields $4.58\% \pm 1.21\%$ misclassifications, i.e. an estimated classification rate of $95.42\% \pm 1.21\%$ (99% confidence interval). This error estimation must be interpreted this way: A classifier that is obtained by the described learning procedure and using the described parameters and training data is estimated to produce an error of $4.58\% \pm 1.21\%$ on unseen data.



(a) (b)

Figure 6.9: Membership functions for the metric variable x_2 and the categorical variable x_6 after training

The final rule base was also discovered in one of the validation cycles. Altogether seven different rule bases were discovered during validation (nine rule bases with two rules, one rule base with four rules). However, most of the other rule bases were very similar and differed only in additionally using the other categorical variable x_3, using x_3 instead of x_2, or using just x_2.

Table 6.6: The confusion matrix of the final classifier obtained by NEFCLASS-J

	Predicted Class							
	malignant		benign		not classified		sum	
malignant	228	(32.62%)	13	(1.86%)	0	(0.00%)	241	(34.99%)
benign	15	(2.15%)	443	(63.38%)	0	(0.00%)	458	(65.01%)
sum	243	(34.76%)	456	(65.24%)	0	(0.00%)	699	(100.00%)

correct: 671 (95.99%), misclassified: 28 (4.01%), error: 60.54.

Table 6.7 compares the result obtained with NEFCLASS-J (last entry) to results obtained with other approaches. The classification performance on unseen data is very good and the classifier is very compact. The error estimates given in the column "Validation" of Table 6.7 are either obtained from 1-leave-out cross validation (discriminant analysis), 10-fold cross validation, or from testing the solution once by withholding 50% of the data for a test set (MLP). Note that the cases with missing values had to be removed only for the MLP. All other approaches are able to handle missing values.

Table 6.7: Comparing the NEFCLASS learning outcome for the WBC data set to some other approaches. Numbers in () are mean values from cross validation. The column "Error" contains an estimated error for unseen data.

Model	Tool	Remarks	Error	Validation
Discriminant Analysis	SPSS	linear model 9 variables	3.95%	1-leave-out
Multilayer Perceptron	SNNS	4 hidden units, RProp	5.18%	50% test set
Decision Tree	C4.5	31 (24.4) nodes, pruned	4.9%	10-fold
Rules from Decision Tree	C4.5rules	8 (7.5) rules using 1–3 variables	4.6%	10-fold
NEFCLASS (metric variables)	NEFCLASS-J (Java version)	2 (2) rules using 1–5 variables	5.86%	10-fold
NEFCLASS (2 categorical variables)	NEFCLASS-J (Java version)	2 (2.1) rules using 1–3 variables	4.58%	10-fold

6.6 Classification as Preprocessing

This section describes an application of NEFCLASS in a Machine Vision Domain [Klose et al., 1999]. In this example the interpretability of the classifier is not important, because it is only to be used for data preprocessing. The purpose of this example is to show that good classification results can be obtained for complex problems even with a relatively simple learning algorithm as used in neuro-fuzzy learning.

In many real world data sets the available training data is more or less unbalanced, i.e. the number of cases of each class can vary dramatically. This is a serious problem for many classification approaches and their learning algorithms. The problem is especially severe, if the classes are not well separated. A typical example is database marketing, where the task of the classifier is to identify prospective buyers who are to receive an offer by mail. A classifier is trained from historical data of "good" customers (who once responded to a mailing) and "bad" customers (who never responded). As response rates of mailings are typically very low (below 5%), there are only few positive examples. Moreover, these cases can be very similar to negative cases and proper separation is hardly possible. In such cases classifiers tend to predict the majority class. This is completely reasonable to minimize the error measure, but does not take into account the special semantics of the problem: the

error is higher, if a good customer is classified as bad than vice versa. A mailing to a bad customer costs no more than the postage, while ignoring a prospective customer means a bigger financial loss. A straightforward way to model this asymmetry is to directly specify the costs of the possible misclassifications.

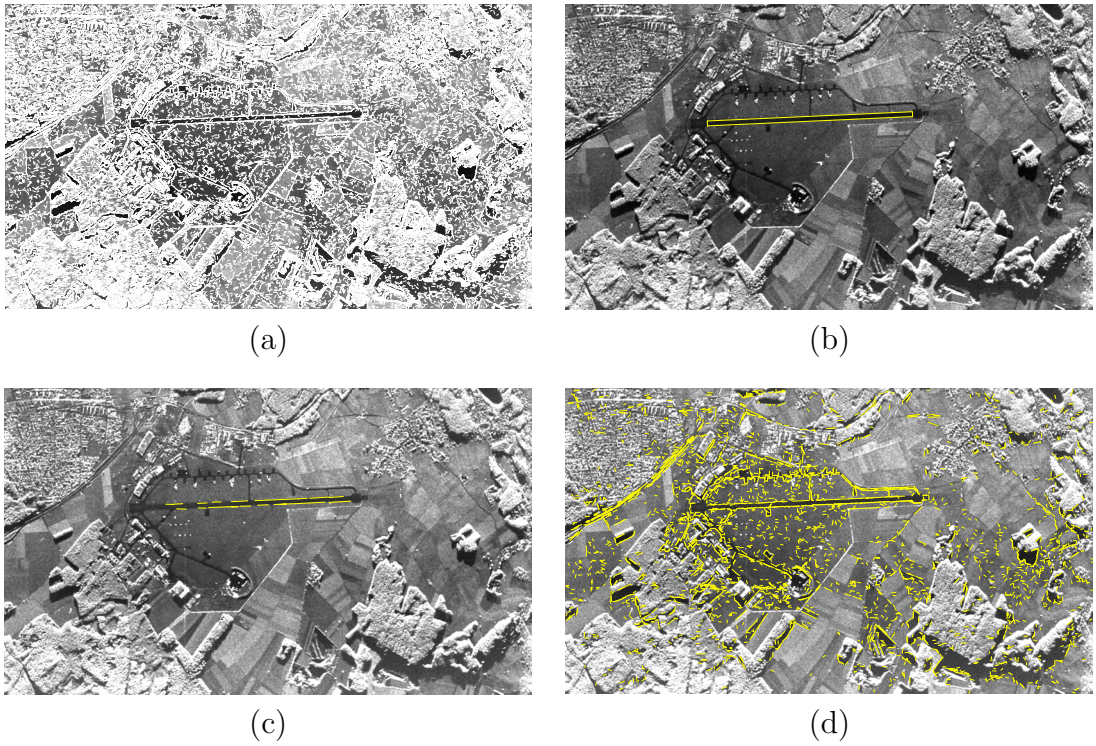
The modified version of the software tool NEFCLASS-X (Unix version) was applied to a machine vision problem which has quite similar characteristics to the customer classification problem explained above, because the error of misclassifying one of the classes is also much larger in this example. In the classification task described in the following too many false negatives can completely prevent the correct recognition of objects, whereas false positives "only" lead to considerably longer execution times.

The considered problem was examined for an industrial project. It deals with the automatic analysis of man-made objects in remotely sensed images. The task is, for example, to detect a runway in an SAR (synthetic aperture radar) image. The detection process is based on edge features which are typical for man-made objects. Figure 6.10(a) shows the result of gradient-based edge detection in an SAR image. These edges are the primitive objects of a subsequent structural analysis. Figure 6.10(b) shows the detected runway. The analysis of the process for this image shows that only 20 lines of about $37,000$ are used to construct this stripe (see Figure 6.10(c)). However, the analysing system must take all of the lines into account. Unfortunately, time consumption is, typically, at least $O(n^2)$. The production process could significantly be speeded up if only the most promising primitive objects are identified and the analysis is started with them.

The idea is to extract features from the image that describe the primitive objects and allow a classifier to decide which lines can be discarded. Each line was described by 11 features. For every image the production process is performed on the complete set of lines. The result is used to divide the lines into those that were used for the construction of complex structures (the *positive* class) and those that were not used (the *negative* class). A classifier has to take into account the special semantics of the task. A classifier that simply always predicts the majority class would have an error rate close to 0% (i.e. only 20 errors out of 37.000 objects in Figure 6.10(a)), but the result would be completely useless and would hinder any object recognition. As a matter of fact, every missed positive can turn out to be very expensive. This has to be considered in the misclassification costs.

For this experiment — whose results were provided by the industrial partner of this project – 17 SAR images depicting 5 different airports were used. Each of the images was analysed in order to detect the runway(s) and the lines were labeled as positive or negative. Four of the 17 images were used as a training set for NEFCLASS-X. The training set contains 253 runway lines and $31,330$ negatives.

The lines from the remaining 13 images were used as test data. The assessment of the classification results takes the specific requirements of the application into account. The ideal classifier for this problem should find all edges which were used to assemble



(a) (b)

(c) (d)

Figure 6.10: (a) 37,659 lines extracted from SAR image (b) detected runway (c) the 20 lines used for the construction of the runway (d) 3,281 lines classified as positive by a modified version of NEFCLASS-X

a runway. On the one hand missing edges can cause the subsequent image analysis to fail. On the other hand the classifier should assign as few negatives as possible to the positive class to reduce the processing time. The behaviour of a classifier can be characterized by a detection and a reduction rate:

$$\begin{aligned} \text{detection rate} &= \frac{\text{correctly detected positives}}{\text{number of positives}}, \\ \text{reduction rate} &= \frac{\text{classified as positive}}{\text{total number of lines}}. \end{aligned}$$

The tool NEFCLASS-X, which was applied in this task, was modified to take into account different misclassification costs for rule creation (rule performance measure) and fuzzy set training [Klose et al., 1999]. In this application the misclassification costs cannot be specified exactly, as the costs of false positives and negatives are barely comparable. The costs of false negatives were empirically set at 300 times the costs of false positives. This results in a detection rate of 97.6% on the training set after learning. The final classifier used 3 fuzzy sets per variable and had a rule base of 601 rules. An interpretation of such a large rule base is obviously infeasible. But this was not in the scope of this experiment anyway.

The averages of the detection and reduction rates on the unseen 13 images are 84% and 17%, respectively. The detection rates on the single images vary from 50% to 100%. Figure 6.10(d) shows the lines NEFCLASS classified as positives in one of the unseen images Apparently this image allows rather high detection and extremely good reduction rates. But also for most of the lower detection rates the higher levels of image analysis were successful, as the missed lines are mainly shorter and less important [Klose et al., 1999].

This example shows, that a neuro-fuzzy approach like NEFCLASS can also be successfully applied in areas where the interpretation of the classifier is not important and only the classification result is of interest. This means that learning algorithms that are mainly designed with interpretable results in mind, do not necessarily cause a decrease in performance. The examples in this chapter show that neuro-fuzzy methods are both useful to obtain interpretable results and to obtain a good performance in data analysis.

Chapter 7

Conclusions

Research in neuro-fuzzy models is less than 10 years old. Most of the early approaches, like the well-known models by Berenji and Jang, concentrated on the area of fuzzy control or on Sugeno-type models where gradient-descent learning can be used [Berenji, 1990, Jang, 1991]. Approaches like ARIC [Berenji, 1992], GARIC [Berenji and Khedkar, 1992], ANFIS [Jang, 1993], or NEFCON [Nauck and Kruse, 1993] were all of this kind. The learning problems considered were mainly function approximation [Wang and Mendel, 1992] or control [Brown and Harris, 1994, Eklund et al., 1992].

Learning fuzzy classifiers from data is a rather new area. Approaches like an early version of NEFCLASS [Nauck and Kruse, 1995] or Fuzzy RuleNet [Tschichold Gürman, 1995] are two of the first models. The interest in learning not only fuzzy sets but also complete fuzzy rule bases increased. These algorithms are usually based on detecting hyperboxes by clustering [Abe and Lan, 1995, Abe et al., 1996] or structure-oriented methods [Simpson, 1992a, Simpson, 1992b, Nauck et al., 1996, Tschichold Gürman, 1997].

Discussions on the interpretability of fuzzy systems and how neuro-fuzzy learning affects the semantics of fuzzy systems are rare [Bersini and Bontempi, 1997b, Bersini et al., 1998, Nauck and Kruse, 1997d, Nauck and Kruse, 1998a, Nauck and Kruse, 1998c]. Usually, the usefulness of neuro-fuzzy approaches is only demonstrated by their performance in prediction. If neuro-fuzzy techniques are used, we must accept that there is a trade-off between precision and readability [Bersini and Bontempi, 1997b, Nauck et al., 1997].

This thesis contributes to the discussion on interpretable learning outcomes by means of neuro-fuzzy methods. We have pointed out that interpretability of a fuzzy system – especially if applied in data analysis – is one of its key advantages. Neuro-fuzzy models can be conveniently used as a link between models for understanding and prediction. To support the readability of a fuzzy model resulting from a training process, one should use approaches that keep the learning algorithms simple – and

therefore understandable – and do not touch the semantics of the underlying fuzzy models. The algorithms in Chapters 4 and 5 conform to this idea.

It was shown that learning algorithms can be defined for successfully creating interpretable fuzzy systems from data. These learning algorithms are based on simple heurisitics that do not require complex computations. We have shown that fuzzy rule learning can be very efficiently done by structure-oriented approaches. These algorithms have clear advantages over cluster-oriented or hyperbox-oriented rule learning procedures. Structure-oriented rule learning algorithms create fuzzy rule bases, which can be easily interpreted. Because they are also very fast, they support the exploratory nature of data analysis. In addition structure-oriented fuzzy rule learning is able to process data that contains numeric and non-numeric attributes and can also easily handle missing values.

The training of membership functions must be constrained in order to obtain an interpretable learning outcome. We have shown that we can define learning algorithms based on the idea of backpropagation, which use an output error to locally compute modifications of parameters in a fuzzy system. The learning algorithms are able to create both accurate and comprehensible solutions for data analysis problems. The neuro-fuzzy learning techniques discussed in this thesis can be readily applied in data analysis or data mining. This was illustrated by some examples using the tool NEFCLASS-J.

In this thesis neuro-fuzzy approaches are viewed as a way to heuristically find parameters of fuzzy models by processing training data with a learning algorithm. Neuro-fuzzy approaches should be seen as development tools that can help to construct a fuzzy model. They are not "automatic fuzzy model creators". The user should always supervise and interpret the learning process. This view matches the exploratory nature of data analysis.

It was shown that although rule weights are a very simple way to create adaptive fuzzy systems, they can destroy the readability of the rule base completely. Rule weights can always be replaced by changes in the fuzzy sets of a rule. However, these changes can lead to non-normal fuzzy sets and to the fact that identical linguistic values are represented in different ways in different rules. Therefore learning rule weights in neuro-fuzzy approaches is not suitable for such data analysis problems, where we need to obtain interpretable models.

If we are only interested in using a neuro-fuzzy model for prediction, we could – from an applicational point of view – ask: why bother with interpretability and semantics? It is important that the model does its job. It is of course possible to leave out all constraints in the learning procedures of a neuro-fuzzy model, consider it only as a convenient tool that can be initialized by prior knowledge and trained by examples, and not look at the final model, as long as it performs to the satisfaction of the user. However, interpretability and clear semantics provide us with obvious advantages like checking the model for plausibility and maintaining it during its life

cycle. These aspects are also important if a model is only to be used for prediction.

In order to applying a neuro-fuzzy learning strategy one more important aspect should be considered: for whatever reason we choose a fuzzy system to solve a problem it cannot be because we need an *optimal* solution. Fuzzy systems are used to exploit the tolerance for suboptimal solutions. So it does not make much sense to select a very sophisticated and expensive training procedure to squeeze the last bit of information from the training data. To do this we must usually forsake the standard fuzzy system architectures but, however, we are confronted with semantical problems instead. The author prefers the view that fuzzy systems are used because they are easy to implement, easy to handle and easy to understand. A learning algorithm to create a fuzzy system from data should also have these features.

Bibliography

S. Abe and M.-S. Lan (1995). *A Method for Fuzzy Rules Extraction Directly from Numerical Data and Its Application to Pattern Classification.* IEEE Trans. Fuzzy Systems, 3(1):18–28.

S. Abe, M.-S. Lan and R. Thawonmas (1996). *Tuning of a Fuzzy Classifier Derived from Data.* Int. J. Approximate Reasoning, 14(1):1–24.

R. Babuska (1998). *Fuzzy Modeling for Control.* Kluwer Academic Publishers, Boston.

H. R. Berenji (1990). *Neural Networks and Fuzzy Logic in Intelligent Control.* In *5th IEEE Int. Symposium on Intelligent Control, Vol. 2*, pages 916–920.

H. R. Berenji (1992). *A Reinforcement Learning-Based Architecture for Fuzzy Logic Control.* Int. J. Approximate Reasoning, 6:267–292.

H. R. Berenji (1998). *Learning and Tuning of Fuzzy Rules.* In H. T. Nguyen and M. Sugeno, eds.: *Fuzzy Systems Modeling and Control*, The Handbooks on Fuzzy Sets, chapter 8, pages 291–310. Kluwer Academic Publishers, Norwell, MA.

H. R. Berenji and P. Khedkar (1992). *Learning and Tuning Fuzzy Logic Controllers through Reinforcements.* IEEE Trans. Neural Networks, 3:724–740.

H. Bersini and G. Bontempi (1997a). *Fuzzy Models Viewed as Multi-Expert Networks.* In M. Mares, R. Mesiar, V. Novak, J. Ramik and A. Stupnanova, eds.: *Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97*, volume II, pages 354–359, Prague. Academia.

H. Bersini and G. Bontempi (1997b). *Now Comes the Time to Defuzzify Neuro-Fuzzy Models.* Fuzzy Sets and Systems, 90:161–170.

H. Bersini, G. Bontempi and M. Birattari (1998). *Is Readability Compatible with Accuracy? From Neuro-Fuzzy to Lazy Learning*. In *Fuzzy-Neuro Systems '98 – Computational Intelligence. Proc. 5th Int. Workshop Fuzzy-Neuro-Systems '98 (FNS'98) in Munich, Germany*, volume 7 of *Proceedings in Artifical Intelligence*, pages 10–25, Sankt Augustin. infix.

H. Bersini, J.-P. Nordvik and A. Bonarini (1993). *A Simple Direct Adaptive Fuzzy Controller Derived from Its Neural Equivalent*. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1993*, pages 345–350, San Francisco.

M. Berthold and D. J. Hand, eds. (1999). *Intelligent Data Analysis: An Introduction*. Springer-Verlag, Berlin. To appear.

M. Berthold and K.-P. Huber (1997). *Tolerating Missing Values in a Fuzzy Environment*. In M. Mares, R. Mesiar, V. Novak, J. Ramik and A. Stupnanova, eds.: *Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97*, volume I, pages 359–362, Prague. Academia.

M. Berthold and K.-P. Huber (1999). *Constructing Fuzzy Graphs from Examples*. Int. J. Intelligent Data Analysis, 3(1). Electronic journal (http://www.elsevier.com/locate/ida).

J. Bezdek (1994). *What is Computational Intelligence?*. In J. Zurada, R. Marks and C. Robinson, eds.: *Computational Intelligence: Imitating Life*, pages 1–12. IEEE Press, Piscataway.

J. C. Bezdek (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, New York.

J. C. Bezdek, E. C.-K. Tsao and N. R. Pal (1992). *Fuzzy Kohonen Clustering Networks*. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, pages 1035–1043, San Diego, CA.

J. Bezdek, J. Keller, R. Krishnapuram and N. Pal (1998). *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. The Handbooks on Fuzzy Sets. Kluwer Academic Publishers, Norwell, MA.

C. Borgelt and R. Kruse (1998). *Attributauswahlmaße für die Induktion von Entscheidungsbäumen*. In G. Nakhaeizadeh, ed.: *Data Mining. Theoretische Aspekte und Anwendungen*, number 27 in *Beiträge zur Wirtschaftsinformatik*, pages 77–98. Physica-Verlag, Heidelberg.

X. Boyen and L. Wehenkel (1999). *Automatic Induction of Fuzzy Decision Trees and its Application to Power System Security Assessment*. Fuzzy Sets and Systems, 102(1):3–19.

L. Breiman, J. Friedman, R. Olsen and C. Stone (1984). *Classification and Regression Trees*. Wadsworth International.

M. Brown and C. Harris (1994). *Neurofuzzy Adaptive Modelling and Control*. Prentice Hall, New York.

J. J. Buckley and E. Eslami (1996). *Fuzzy Neural Networks: Capabilities*. In W. Pedrycz, ed.: *Fuzzy Modelling: Paradigms and Practice*, pages 167–183. Kluwer, Boston.

J. J. Buckley and Y. Hayashi (1994). *Fuzzy Neural Networks: A Survey*. Fuzzy Sets and Systems, 66:1–13.

J. J. Buckley and Y. Hayashi (1995). *Neural Networks for Fuzzy Systems*. Fuzzy Sets and Systems, 71:265–276.

A. Dempster, N. Laird and D. Rubin (1977). *Maximum Likelihood from Incomplete Data via the EM algorithm*. Journal of the Royal Statistic Society, Series B, 39:1–38.

H. L. Dreyfus (1979). *What Computers Can't Do: The Limits of Artificial Intelligence*. Harper & Row, New York.

D. Dubois, J. Lang and H. Prade (1989). *Automated Reasoning Using Possibilistic Logic: Semantics, Belief Revision and Variable Certainty Weights*. In *Proc. 5th Workshop on Uncertainty in Artificial Intelligence*, pages 81–87, Ontario. Windsor.

D. Dubois and H. Prade (1988). *Possibility Theory*. Plenum Press, New York.

P. Eklund, F. Klawonn and D. Nauck (1992). *Distributing Errors in Neural Fuzzy Control*. In *Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks (IIZUKA'92)*, pages 1139–1142.

S. Fahlman and C. Lebiere (1990). *The Cascade Correlation Learning Architecture*. In D. Touretzky, ed.: *Advances in Neural Information Processing 2*, pages 524–532. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

M. Fayyad, Usama, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy, eds. (1996). *Advances in Knowledge Discovery and Data Mining*. MIT Press, Menlo Park, CA.

T. Feuring and W. M. Lippe (1996). *Learning in Fuzzy Neural Networks*. In *Proc. IEEE Int. Conf. on Neural Networks 1996 (ICNN'96)*, pages 1061–1066, Washington.

R. Fisher (1936). *The Use of Multiple Measurements in Taxonomic Problems*. Annual Eugenics, 7(Part II):179–188.

K. Funahashi (1989). *On the Approximate Realization of Continuous Mappings by Neural Networks*. Neural Networks, 2:183–192.

J. Gebhardt and R. Kruse (1994a). *Focusing and Learning in Possibilistic Dependencyy Networks*. In *Postproceedings of 2nd Gauss Symposium (Conference B: Statistical Sciences)*, Berlin. De Gruyter.

J. Gebhardt and R. Kruse (1994b). *Learning Possibilistic Networks from Data*. In *Proc. of Fifth Int. Workshop on Artificial Intelligence and Statistics*, pages 233–244, Fort Lauderdale, Florida.

A. Grauel, G. Klene and L. Ludwig (1997). *Data Analysis by Fuzzy Clustering Methods*. In A. Grauel, W. Becker and F. Belli, eds.: *Fuzzy-Neuro-Systeme'97 – Computational Intelligence. Proc. 4th Int. Workshop Fuzzy-Neuro-Systeme '97 (FNS'97) in Soest, Germany*, Proceedings in Artificial Intelligence, pages 563–572, Sankt Augustin. infix.

A. Grauel and L. Ludwig (1999). *Construction of Differentiable Membership Functions*. Fuzzy Sets and Systems, 101(2):219–225.

A. Grauel and Mackenberg (1997). *Mathematical Analysis of the Sugeno Controller Leading to General Design Rules*. Fuzzy Sets and Systems, 85(2):165–175.

N. Tschichold Gürman (1995). *Generation and Improvement of Fuzzy Classifiers with Incremental Learning Using Fuzzy RuleNet*. In K. M. George, J. H. Carrol, E. Deaton, D. Oppenheim and J. Hightower, eds.: *Applied Computing 1995. Proc. 1995 ACM Symposium on Applied Computing, Nashville, Feb. 26–28*, pages 466–470, New York. ACM Press.

N. Tschichold Gürman (1996). *RuleNet – A New Knowledge-Based Artificial Neural Network Model with Application Examples in Robotics*. PhD thesis, ETH Zürich.

N. Tschichold Gürman (1997). *The Neural Network Model RuleNet and its Application to Mobile Robot Navigation*. Fuzzy Sets and Systems, 85:287–303.

J. F. Hair, R. E. Anderson, R. L. Tatham and W. C. Black (1998). *Multivariate Data Analysis*. Prentice-Hall, Upper Saddle River, NJ, Fifth Edition edition.

S. K. Halgamuge (1995). *Advanced Methods for Fusion of Fuzzy Systems and Neural Networks in Intelligent Data Processing*. PhD thesis, Technische Hochschule Darmstadt.

S. K. Halgamuge and M. Glesner (1994). *Neural Networks in Designing Fuzzy Systems for Real World Applications*. Fuzzy Sets and Systems, 65:1–12.

D. J. Hand (1998). *Intelligent Data Analysis: Issues and Opportunities*. Int. J. Intelligent Data Analysis, 2(2). Electronic journal (http://www.elsevier.com/locate/ida).

S. Haykin (1994). *Neural Networks. A Comprehensive Foundation*. Macmillan College Publishing Company, New York.

D. O. Hebb (1949). *The Organization of Behavior*. Wiley, New York.

R. Hecht-Nielsen (1989). *Theory of the Back-Propagation Neural Network*. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN'89), Vol. 1*, pages 593–606.

R. Hecht-Nielsen (1990). *Neurocomputing*. Addison-Wesley, Reading, MA.

D. Heckerman (1988). *Probabilistic Interpretation for MYCIN's Certainty Factors*. In J. Lemmer and L. Kanal, eds.: *Uncertainty in Artificial Intelligence (2)*, pages 167–196. North-Holland, Amsterdam.

C. Higgins and R. Goodman (1993). *Learning Fuzzy Rule-based Neural Networks for Control*. Advances in Neural Information Processing Systems, 5:350–357.

J. Hopf and F. Klawonn (1994). *Learning the Rule Base of a Fuzzy Controller by a Genetic Algorithm*. In [Kruse et al., 1994b], pages 63–73, Braunschweig. Vieweg.

F. Höppner, F. Klawonn, R. Kruse and T. Runkler (1999). *Fuzzy Cluster Analysis*. Wiley, Chichester.

M. Hornik, M. Stinchcombe and H. White (1989). *Multilayer Feedfoward Networks Are Universal Approximators*. Neural Networks, 2:359–366.

M. Hornik, M. Stinchcombe and H. White (1990). *Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks*. Neural Networks, 3:551–560.

H. Ichihashi, T. Shirai, K. Nagasaka and T. Miyoshi (1996). *Neuro-Fuzzy ID3: A Method of Inducing Fuzzy Decision Trees with Linear Programming for Maximizing Entropy and an Algebraic Method for Incremental Learning*. Fuzzy Sets and Systems, 81(1):157–167.

W. Inmon (1996). *Building the Data Warehouse*. Wiley, New York.

J. S. R. Jang (1991). *Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm*. In *Proc. Ninth National Conf. on Artificial Intelligence (AAAI-91)*, pages 762–767.

J. S. R. Jang (1993). *ANFIS: Adaptive-Network-Based Fuzzy Inference Systems*. IEEE Trans. Systems, Man & Cybernetics, 23:665–685.

J.-S. R. Jang and C.-T. Sun (1993). *Functional Equivalence Between Radial Basis Function Networks and Fuzzy Inference Systems*. IEEE Trans. Neural Networks, 4:156–163.

J.-S. Jang, C. Sun and E. Mizutani (1997). *Neuro Fuzzy and Soft Computing*. Prentice Hall, Upper Saddle River, NJ.

C. Z. Janikow (1996). *Exemplar based Learning in Fuzzy Decision Trees*. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1996*, pages 1500–1505, New Orleans.

C. Z. Janikow (1998). *Fuzzy Decision Trees: Issues and Methods*. IEEE Trans. Systems, Man & Cybernetics. Part B: Cybernetics, 28(1):1–14.

L. P. Kaelbling, M. H. Littman and A. W. Moore (1996). *Reinforcement Learning: A Survey*. J. Artificial Intelligence Research, 4:237–285.

A. Keller and F. Klawonn (1998). *Generating Classification Rules by Grid Clustering*. In *Proc. Third European Workshop on Fuzzy Decision Analysis and Neural Networks for Management, Planning, and Optimization (EFDAN'98)*, pages 113–121, Dortmund.

J. M. Keller and H. Tahani (1992). *Backpropagation Neural Networks for Fuzzy Logic*. Information Sciences, 62:205–221.

J. Kinzel, F. Klawonn and R. Kruse (1994). *Modifications of Genetic Algorithms for Designing and Optimizing Fuzzy Controllers*. In *Proc. IEEE Conference on Evolutionary Computation*, pages 28–33, Orlando, FL. IEEE.

F. Klawonn, J. Gebhardt and R. Kruse (1995a). *Fuzzy Control on the Basis of Equality Relations with an Example from Idle Speed Control*. IEEE Trans. Fuzzy Systems, 3(3):336–350.

F. Klawonn and A. Keller (1997). *Fuzzy Clustering and Fuzzy Rules*. In M. Mares, R. Mesiar, V. Novak, J. Ramik and A. Stupnanova, eds.: *Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97*, volume I, pages 193–197, Prague. Academia.

F. Klawonn and R. Kruse (1995). *Clustering Methods in Fuzzy Control*. In W. Gaul and D. Pfeifer, eds.: *From Data to Knowledge: Theoretical and Practical Aspects of Classification, Data Analysis and Knowledge Organization*, pages 195–202. Springer-Verlag, Berlin.

F. Klawonn and R. Kruse (1997). *Constructing a Fuzzy Controller from Data*. Fuzzy Sets and Systems, 85:177–193.

F. Klawonn, D. Nauck and R. Kruse (1995b). *Generating Rules from Data by Fuzzy and Neuro-Fuzzy Methods*. In *Proc. Fuzzy-Neuro-Systeme'95*, pages 223–230, Darmstadt.

A. Klose, D. Nauck, K. Schulz and U. Thönessen (1999). *Learning a Neuro-Fuzzy Classifier from Unbalanced Data in a Machine Vision Domain*. In *Proc. 6th Int. Workshop on Fuzzy-Neuro Systems 1999 (FNS'99)*, pages 133–144, Leipzig. Leipziger Universitätsverlag.

A. Klose, A. Nürnberger and D. Nauck (1998). *Some Approaches to Improve the Interpretability of Neuro-Fuzzy Classifiers*. In *Proc. Sixth European Congress on Intelligent Techniques and Soft Computing (EUFIT98)*, pages 629–633, Aachen.

T. Kohonen (1984). *Self-Organization and Associative Memory*. Springer-Verlag, Berlin.

I. Kononenko (1995). *On Biases in Estimating Multi-valued Attributes*. In *Proc. 1st International Conference on Knowledge Discovery and Data Mining*, pages 1034–1040, Montreal.

B. Kosko (1992). *Neural Networks and Fuzzy Systems. A Dynamical Systems Approach to Machine Intelligence*. Prentice Hall, Englewood Cliffs, NJ.

A. Krone and H. Kiendl (1996). *Rule-based Decision Analysis with FUZZY-ROSA method*. In *Proc. First European Workshop on Fuzzy Decision Analysis and Neural Networks for Management, Planning, and Optimization (EFDAN'96)*, pages 109–114, Dortmund.

R. Kruse, J. Gebhardt and F. Klawonn (1994a). *Foundations of Fuzzy Systems*. Wiley, Chichester.

R. Kruse, J. Gebhardt and R. Palm, eds. (1994b). *Fuzzy Systems in Computer Science*. Vieweg, Braunschweig.

R. Kruse and K. D. Meyer (1987). *Statistics with Vague Data*. Reidel, Dordrecht.

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

Y. Le Cun, J. Denker and S. S. (1990). *Optimal Brain Damage*. In *Advances in Neural Information Processing Systems, NIPS'89*, volume 2, pages 589–605, San Mateo, CA. Morgan Kaufmann.

M. Lee and H. Takagi (1993). *Integrating Design Stages of Fuzzy Systems Using Genetic Algorithms*. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1993*, pages 612–617, San Francisco.

C.-T. Lin and C. S. G. Lee (1996). *Neural Fuzzy Systems. A Neuro-Fuzzy Synergism to Intelligent Systems*. Prentice Hall, New York.

E. H. Mamdani and S. Assilian (1975). *An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller*. Int. J. Man Machine Studies, 7:1–13.

W. S. McCulloch and W. Pitts (1943). *A Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematical Biophysics, 5:115–133.

M. L. Minsky and S. Papert (1969). *Perceptrons*. MIT Press, Cambridge, MA.

T. M. Mitchell (1997). *Machine Learning*. McGraw-Hill, New York, NY.

S. Mitra and L. Kuncheva (1995). *Improving Classification Performance Using Fuzzy MLP and Two-Level Selective Partitioning of the Feature Space*. Fuzzy Sets and Systems, 70:1–13.

G. Nakhaeizadeh (1998). *Wissensentdeckung in Datenbanken und Data Mining: Ein Überblick*. In G. Nakhaeizadeh, ed.: *Data Mining. Theoretische Aspekte und Anwendungen*, number 27 in *Beiträge zur Wirtschaftsinformatik*, pages 1–33. Physica-Verlag, Heidelberg.

D. Nauck (1994a). *Building Neural Fuzzy Controllers with NEFCON-I*. In R. Kruse, J. Gebhardt and R. Palm, eds.: *Fuzzy Systems in Computer Science*, pages 141–151. Vieweg, Braunschweig.

D. Nauck (1994b). *Modellierung Neuronaler Fuzzy-Regler*. PhD thesis, Technische Universität Braunschweig.

D. Nauck and F. Klawonn (1996). *Neuro-Fuzzy Classification Initialized by Fuzzy Clustering*. In *Proc. Fourth European Congress on Intelligent Techniques and Soft Computing (EUFIT96)*, pages 1551–1555, Aachen. Verlag und Druck Mainz.

D. Nauck, F. Klawonn and R. Kruse (1997). *Foundations of Neuro-Fuzzy Systems*. Wiley, Chichester.

D. Nauck and R. Kruse (1993). *A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation*. In *Proc. IEEE Int. Conf. on Neural Networks 1993*, pages 1022–1027, San Francisco.

D. Nauck and R. Kruse (1994). *NEFCON-I: An X-Window Based Simulator for Neural Fuzzy Controllers*. In *Proc. IEEE Int. Conf. Neural Networks 1994 at IEEE WCCI'94*, pages 1638–1643, Orlando, FL.

D. Nauck and R. Kruse (1995). *NEFCLASS – A Neuro-Fuzzy Approach for the Classification of Data.* In K. M. George, J. H. Carrol, E. Deaton, D. Oppenheim and J. Hightower, eds.: *Applied Computing 1995. Proc. 1995 ACM Symposium on Applied Computing, Nashville, Feb. 26–28*, pages 461–465. ACM Press, New York.

D. Nauck and R. Kruse (1996). *Designing Neuro-Fuzzy Systems Through Backpropagation.* In W. Pedrycz, ed.: *Fuzzy Modelling: Paradigms and Practice*, pages 203–228. Kluwer, Boston.

D. Nauck and R. Kruse (1997a). *Function Approximation by NEFPROX.* In *Proc. Second European Workshop on Fuzzy Decision Analysis and Neural Networks for Management, Planning, and Optimization (EFDAN'97)*, pages 160–169, Dortmund.

D. Nauck and R. Kruse (1997b). *A Neuro-Fuzzy Method to Learn Fuzzy Classification Rules from Data.* Fuzzy Sets and Systems, 89:277–288.

D. Nauck and R. Kruse (1997c). *Neuro-Fuzzy Systems for Function Approximation.* In A. Grauel, W. Becker and F. Belli, eds.: *Fuzzy-Neuro-Systeme'97 – Computational Intelligence. Proc. 4th Int. Workshop Fuzzy-Neuro-Systeme '97 (FNS'97) in Soest, Germany*, Proceedings in Artificial Intelligence, pages 316–323, Sankt Augustin. infix.

D. Nauck and R. Kruse (1997d). *What are Neuro-Fuzzy Classifiers?.* In M. Mares, R. Mesiar, V. Novak, J. Ramik and A. Stupnanova, eds.: *Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97*, volume III, pages 228–233, Prague. Academia.

D. Nauck and R. Kruse (1998a). *How the Learning of Rule Weights Affects the Interpretability of Fuzzy Systems.* In *Proc. IEEE Int. Conf. on Fuzzy Systems 1998*, pages 1235–1240, Anchorage.

D. Nauck and R. Kruse (1998b). *NEFCLASS-X – A Soft Computing Tool to Build Readable Fuzzy Classifiers.* BT Technology Journal, 16(3):180–190.

D. Nauck and R. Kruse (1998c). *A Neuro-Fuzzy Approach to Obtain Interpretable Fuzzy Systems for Function Approximation.* In *Proc. IEEE Int. Conf. on Fuzzy Systems 1998*, pages 1106–1111, Anchorage.

D. Nauck and R. Kruse (1999a). *Neuro–Fuzzy Systems for Function Approximation.* Fuzzy Sets and Systems, 101:261–271.

D. Nauck and R. Kruse (1999b). *Obtaining Interpretable Fuzzy Classification Rules from Medical Data.* Artificial Intelligence in Medicine. Accepted.

D. Nauck, U. Nauck and R. Kruse (1996). *Generating Classification Rules with the Neuro-Fuzzy System NEFCLASS*. In *Proc. Biennial Conference of the North American Fuzzy Information Processing Society NAFIPS'96*, pages 466–470, Berkeley, CA.

D. Nauck, U. Nauck and R. Kruse (1999). *NEFCLASS for JAVA – New Learning Algorithms*. In *Proc. 18th International Conf. of the North American Fuzzy Information Processing Society (NAFIPS99)*, pages 472–476, New York, NY. IEEE.

R. Neuneier and H.-G. Zimmermann (1998). *How to train neural networks*. In *Tricks of the Trade: How to Make Algorithms Really Work*, LNCS State-of-the-Art-Survey. Springer-Verlag, Berlin.

H. Nomura, I. Hayashi and N. Wakami (1992). *A Learning Method of Fuzzy Inference Rules by Descent Method*. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, pages 203–210, San Diego, CA.

A. Nürnberger, D. Nauck and R. Kruse (1999). *Neuro-Fuzzy Control Based on the NEFCON-Model: Recent Developments*. Soft Computing, 2(4):168–182.

S. K. Pal and S. Mitra (1992). *Multi-layer Perceptron, Fuzzy Sets and Classification*. IEEE Trans. Neural Networks, 3:683–697.

J. Pearl (1988). *Probabilistic Reasoning in Intelligent Systems. Networks of Plausible Inference*. Morgan Kaufmann, San Francisco.

W. Pedrycz and H. C. Card (1992). *Linguistic Interpretation of Self-Organizing Maps*. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, pages 371–378, San Diego, CA.

T. Poggio and F. Girosi (1989). *A theory of networks for approximation and learning*. A.I. Memo 1140, MIT, Cambridge, MA.

J. Quinlan (1986). *Induction of Decision Trees*. Machine Learning, 1:81–106.

J. Quinlan (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo, CA.

J. Rissanen (1983). *A Universal Prior for Integers and Estimation by Minimum Description Length*. Annals of Statistic, 11:416–431.

R. Rojas (1996). *Neural Networks – A Systematic Introduction*. Springer-Verlag, Berlin.

F. Rosenblatt (1958). *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain*. Psychological Review, 65:386–408.

F. Rosenblatt (1962). *Principles of Neurodynamics*. Spartan Books, New York.

D. E. Rumelhart, G. E. Hinton and R. J. Williams (1986a). *Learning Internal Representations by Error Propagation*. In [Rumelhart and McClelland, 1986], pages 318–362. MIT Press, Cambridge, MA.

D. E. Rumelhart, G. E. Hinton and R. J. Williams (1986b). *Learning Representations by Back-Propagating Errors*. Nature, 323:533–536.

D. E. Rumelhart and J. L. McClelland, eds. (1986). *Parallel Distributed Processing: Explorations in the Microstructures of Cognition. Foundations*, volume 1. MIT Press, Cambridge, MA.

S. Shao (1988). *Fuzzy Self-Organizing Controller and Its Application for Dynamic Processes*. Fuzzy Sets and Systems, 26:151–164.

S. Siekmann, R. Neuneier, H. G. Zimmermann and R. Kruse (1999). *Neuro-Fuzzy Methods Applied to the German Stock Index DAX*. In R. Ribero, H.-J. Zimmermann, R. Yager and J. Kacprzyk, eds.: *Soft Computing in Financial Engineering*. Physica-Verlag, Heidelberg. To appear.

P. K. Simpson (1992a). *Fuzzy Min-Max Neural Networks – Part 1: Classification*. IEEE Trans. Neural Networks, 3:776–786.

P. K. Simpson (1992b). *Fuzzy Min-Max Neural Networks – Part 2: Clustering*. IEEE Trans. Fuzzy Systems, 1:32–45.

M. Sugeno (1985). *An Introductory Survey of Fuzzy Control*. Information Sciences, 36:59–83.

M. Sugeno and T. Yasukawa (1993). *A Fuzzy-Logic-Based Approach to Qualitative Modeling*. IEEE Trans. Fuzzy Systems, 1:7–31.

H. Takagi and M. Lee (1993). *Neural Networks and Genetic Algorithms*. In E. P. Klement and W. Slany, eds.: *Fuzzy Logic in Artificial Intelligence (FLAI93)*, pages 68–79, Berlin. Springer-Verlag.

T. Takagi and M. Sugeno (1985). *Fuzzy Identification of Systems And Its Applications to Modeling and Control*. IEEE Trans. Systems, Man & Cybernetics, 15:116–132.

H. Timm and F. Klawonn (1998). *Classification of Data with Missing Values*. In *Proc. Sixth European Congress on Intelligent Techniques and Soft Computing (EUFIT98)*, pages 639–644, Aachen.

H. Timm and R. Kruse (1998). *Fuzzy Cluster Analysis with Missing Values*. In *Proc. 17th International Conf. of the North American Fuzzy Information Processing Society (NAFIPS98)*, pages 242–246, Pensacola, FL.

P. Vuorimaa (1994). *Fuzzy Self-Organizing Map*. Fuzzy Sets and Systems, 66:223–231.

L.-X. Wang and J. M. Mendel (1991). *Generation Rules by Learning from Examples*. In *International Symposium on Intelligent Control*, pages 263–268. IEEE Press.

L.-X. Wang and J. M. Mendel (1992). *Generating fuzzy rules by learning from examples*. IEEE Trans. Syst., Man, Cybern., 22(6):1414–1427.

P. J. Werbos (1974). *Beyond Regressions: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA.

H. White (1990). *Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings*. Neural Networks, 3:535–549.

B. Widrow and M. E. Hoff (1960). *Adaptive Switching Circuits*. In *IRE WESCON Convention Record*, pages 96–104, New York. IRE.

W. Wolberg and O. Mangasarian (1990). *Multisurface Method of Pattern Separation for Medical Diagnosis Applied to Breast Cytology*. Proc. National Academy of Sciences, 87:9193–9196.

Y. Yuan and M. J. Shaw (1995). *Induction of Fuzzy Decision Trees*. Fuzzy Sets and Systems, 69(2):125–139.

L. A. Zadeh (1965). *Fuzzy Sets*. Information and Control, 8:338–353.

L. A. Zadeh (1994a). *Fuzzy Logic, Neural Networks and Soft Computing*. Communications of the ACM, 37(3):77–84.

L. A. Zadeh (1994b). *Soft Computing and Fuzzy Logic*. IEEE Software, 11(6):48–56.

L. A. Zadeh (1996). *Fuzzy Logic and the Calculi of Fuzzy Rules and Fuzzy Graphs: A Precis*. Int. J. Multiple-Valued Logic, 1:1–38.

A. Zell (1994). *Simulation Neuronaler Netze*. Addision-Wesley, Bonn.

H. G. Zimmermann, R. Neuneier, H. Dichtl and S. Siekmann (1996). *Modeling the German Stock Index DAX with Neuro-Fuzzy*. In *Proc. Fourth European Congress on Intelligent Techniques and Soft Computing (EUFIT96)*, Aachen. Verlag und Druck Mainz.

H. J. Zimmermann (1996). *Fuzzy Set Theory and its Applications*, 3rd edition. Kluwer Academic Publishers, Boston.

J. M. Zurada (1992). *Introduction to Artificial Neural Systems*. West Publishing Company, St. Paul, MN.

Index

List of Symbols

$\mathbb{1}_M$	characteristic function of set M
a_u	activation of network unit u
A_u	activation function of network unit u
$\mathrm{ant}(R)$	antecedent of fuzzy rule R
$\mathrm{con}(R)$	consequent of fuzzy rule R
ex_u	external input of network unit u
$\mathcal{E}$	fuzzy rule error
$\mathcal{F}(X)$	set of all fuzzy sets of X
$\mathcal{F}$	short form of $\mathcal{F}(\mathbb{R})$
$F_{\mathcal{R}}$	fuzzy system with rule base $\mathcal{R}$
$\mathcal{L}$	free learning problem
$\tilde{\mathcal{L}}$	fixed learning problem
μ, ν	fuzzy sets, membership functions
$[\mu]_\alpha$	α-cut of μ
$MF_{\mathcal{R}}$	Mamdani-type fuzzy system with fuzzy rule base $\mathcal{R}$
net_u	network input of network unit u
NET_u	propagation function of network unit u
o_u	output of network unit u
O_u	output function of network unit u
$\mathbb{R}$	the set of all real values
$\mathcal{R}$	rule base of a fuzzy system
R	fuzzy rule, rule unit
$SF_{\mathcal{R}}$	Sugeno-type fuzzy system with fuzzy rule base $\mathcal{R}$
τ	degree of fulfilment of a fuzzy rule
U	set of network unit
U_I	set of input units of a network
U_H	set of hidden units of a network
U_O	set of output units of a network
W	network structure
$W(u, v)$	connection weight between network units u and v

Curriculum Vitae

Personal Data

Name:	Detlef Nauck
Date of birth:	March 20, 1964
Place of birth:	Braunschweig, Germany
Marital status:	married to Ulrike Nauck, nee Harder

Education

1970 – 1983	School education
1983	Final examination
1985 – 1990	Studies of computer science and business administration at the Technical University of Braunschweig, Germany
1990	Master degree in computer science (Diplom-Informatiker)
1994	Ph.D. in computer science (Dr. rer. nat., summa cum laude) from the Technical University of Braunschweig, Germany

Employment

1983 – 1985	German Federal Armed Forces, reserve officer
1987 – 1990	Research Assistant for software development, Department of Mechanical Engineering, Technical University of Braunschweig, Germany
1990 – 1994	Researcher, Department of Computer Science, Technical University of Braunschweig, Braunschweig, Germany
1994 – 1996	Senior Researcher and Lecturer (Postdoc), Department of Computer Science, Technical University of Braunschweig, Germany
1991 – 1999	Lecturer for Computer Science at Fachhochschule Braunschweig-Wolfenbüttel, FB Versorgungstechnik, Wolfenbüttel, Germany
1996 – 1999	Senior Research Fellow and Senior Lecturer, Faculty of Computer Science, Otto-von-Guericke University of Magdeburg, Germany
since October 1999	Senior Research Scientist, Intelligent Systems Research Group, BT Labs, Adastral Park, Martlesham Heath, Ipswich, United Kingdom

Magdeburg, 01.03.2000 *Detlef Nauck*