# Using Domain-Specific Basic Functions for the Analysis of Supervised Artificial Neural Networks

*Berend Jan van der Zwaag*

Acknowledgment:
This work has been made possible with support from the Centre for Telematics and Information Technology (CTIT)

Cover design:
The background image shown on the cover is a 43-neighbor dihedral tiling discovered by Berend Jan van der Zwaag; this is a tiling of the plane using two different tiles, where each tile has exactly 43 neighboring tiles [E. Friedman and B.J. van der Zwaag (2002), "Constant neighbor dihedral tilings with 15, 32, and 43 neighbors," *Geombinatorics*, vol. XI, no. 3 (Jan.), pp. 74-77].
The illustration on the front cover was drawn by Irene van der Zwaag-Tong, who also made the drawing of a biological neuron shown on page 8.

Zwaag, Berend Jan van der

Using domain-specific basic functions for the analysis of supervised artificial neural networks

(Nederlandse titel: Domeinafhankelijke basisfuncties voor de analyse van gesuperviseerde kunstmatige neurale netwerken)

ISBN: 90-365-2008-8

# USING DOMAIN-SPECIFIC BASIC FUNCTIONS FOR THE ANALYSIS OF SUPERVISED ARTIFICIAL NEURAL NETWORKS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof.dr. F.A. van Vught,
volgens besluit van het College van Promoties
in het openbaar te verdedigen
op woensdag 17 december 2003 om 15.00 uur

door

Berend Jan van der Zwaag
geboren op 26 november 1969
te Dokkum

Dit proefschrift is goedgekeurd door de promotor,

prof.dr.ir. C.H. Slump

# Contents

# Voorwoord

Sinds mijn eerste ervaringen op het gebied van kunstmatige neurale netwerken ben ik gefascineerd geraakt door de complexiteit die verscholen ligt achter de eenvoud van hun bouwstenen. Waar anderen worden geremd door deze complexiteit die een heldere kijk op de door neurale netwerken verworven kennis verhindert, vond ik het juist een aanmoediging om op zoek te gaan naar sleutels die de magische doos kunnen openen en de inhoud bloot kunnen leggen.

Door de jaren heen heb ik verschillende interessante zijwegen bewandeld door het landschap van de computationele intelligentie, gebruik makend van technieken zoals genetische algoritmen en fuzzy logica, om uiteindelijk weer terug te keren naar het zoeken naar begrijpelijke kennisextractie uit feedforward-type kunstmatige neurale netwerken. Dit proefschrift is het resultaat van dat onderzoek. Dit betekent echter niet dat er niets meer valt te onderzoeken. Integendeel, zolang kunstmatige neurale netwerken toegepast blijven worden in een immer uitdijende wereld van toepassingen, blijven er volop nieuwe wegen te bewandelen. Ik hoop met het in dit proefschrift gepresenteerde werk een bijdrage te leveren aan dit interessante onderzoeksgebied. De nieuwe kennisextractietechnieken die in dit proefschrift zijn beschreven kunnen ontwikkelaars van toepassingen van neurale netwerken helpen bij het ontwerpen van gemakkelijker te begrijpen kunstmatige neurale netwerksystemen.

Ik ben veel mensen dank verschuldigd zonder wie dit werk niet zou zijn geweest wat het nu is. Enkelen van hen wil ik in het bijzonder bedanken, in een min of meer chronologische volgorde: Leo Veelenturf, die mij inleidde in de wereld van de kunstmatige neur*on*ale netwerken; Lakhmi Jain en mijn voormalige collega's in het Knowledge-Based Intelligent Engineering Systems Centre en in het Advanced Computing Research Centre binnen de University of South Australia, voor de vrijheid die zij mij gaven om het gebied van de computationele intelligentie verder te verkennen en voor de gelegenheid die ik daardoor kreeg om actief betrokken te zijn bij de verspreiding van de nieuwste ontwikkelingen, zowel theoretisch als in praktische toepassingen, aan een breder publiek; Otto Herrmann, die mij actief betrokken gemaakt heeft bij de kennisoverdracht aan het regionale midden- en kleinbedrijf in het kader van het EUREGIO Computational Intelligence Centre; Sabih Gerez en mijn andere collega's in het ECIC, door wie ik me weer thuis ben gaan voelen binnen de Universiteit Twente na meer dan drie jaren *downunder*; mijn promotor Kees Slump voor zijn welwillendheid om mij te begeleiden in het onderzoek dat uiteindelijk geleid heeft tot dit proefschrift; Ben Spaanenburg voor de inzichtelijke discussies en het laten zien dat er verschillende oplossingen voor hetzelfde probleem kunnen zijn, en verschillende problemen dezelfde oplossing kunnen hebben; en Anneke van Essen-Rekers voor haar waardevolle assistentie waardoor ik mij kon concentreren op mijn onderzoek.

Verder wil ik graag mijn collega's en de studenten in de leerstoel Signalen en Systemen bedanken voor de prettige werksfeer gedurende de afgelopen vier jaar. Dit proefschrift zou ook niet mogelijk zijn geweest zonder de steun en het begrip van mijn familie en vrienden. In het bijzonder wil ik mijn vrouw Irene bedanken voor haar voortdurende geduld en begrip. En tot slot dank ik God voor deze wereld waarin iedere dag weer nieuwe dingen brengt, waarmee het leven op aarde interessant blijft.

Berend Jan van der Zwaag
Hengelo (O), november 2003

# Preface

Since my first experiences in the field of artificial neural networks I have been intrigued by the complexity that lies hidden beneath the simplicity of their building blocks. Whereas others might be held back by that complexity that prevents a clear look into the knowledge acquired by neural networks through training, I found it encouraging to search and discover keys that would open the magic box and reveal its contents.

As the years passed, I explored several interesting deviations through the landscape of computational intelligence, using techniques such as genetic algorithms and fuzzy logic control, eventually returning to the investigation into comprehensible knowledge extraction from feedforward-type artificial neural networks. This thesis is the result of that investigation. Even so, this does not mean that there is nothing left to investigate. To the contrary, as long as artificial neural networks continue to be used in an ever expanding world of applications, there will be plenty of new routes to explore. With the work presented in this thesis, I hope to deliver a contribution to this interesting field. The new knowledge extraction techniques described in this thesis can assist neural network application designers in designing more easily comprehensible artificial neural network systems.

I am indebted to many people without whom this work would not have been what it is now. A few of them I want to thank in particular, in a more or less chronological order: Leo Veelenturf for introducing me to the field of artificial neural networks; Lakhmi Jain and my former colleagues in the Knowledge-Based Intelligent Engineering Systems Centre and in the Advanced Computing Research Centre in the University of South Australia, for giving me the freedom to further explore the field of computational intelligence and for providing me with the opportunity to be actively involved in bringing to a wider audience the state of the art of theoretical developments and a wide range of applications; Otto Herrmann for getting me actively involved in the knowledge transfer to small and medium-sized regional companies in the framework of the EUREGIO Computational Intelligence Centre; Sabih Gerez and my other colleagues in the ECIC for making me feel at home again in the University of Twente after more than three years downunder; my promotor Kees Slump for his willingness to guide me in the research that eventually led to this thesis; Ben Spaanenburg for the insightful discussions and for showing me that there can be different solutions to the same problem and different problems with the same solution; and Anneke van Essen-Rekers for her invaluable assistance allowing me to concentrate on my research.

I further wish to thank my colleagues and the students in the Signals and Systems group for their technical assistance and the fine working ambience during the past four years. This thesis would also have been impossible without the support and understanding from my relatives and friends. In particular I want to express my gratitude to my wife Irene for her continuous patience and understanding. And last but not least I thank God for this world in which new things can be explored and experienced every day, which keeps life on earth interesting.

Berend Jan van der Zwaag
Hengelo (O), November 2003

# Chapter 1

# Introduction

In the past 20 years artificial neural networks have gained renewed popularity as "universal problem solvers" (Siegelmann 1998, Haykin 1994, p. 39). Thousands of articles are published on the subject every year (a recent bibliography by Oja et al. (2003) lists more than two thousand papers from 1998-2001 on the topic of Self-Organizing Maps alone). However, one of the major weaknesses of neural networks is their inexplicability (Andrews et al. 1995). This is an important aspect of the functionality of any technology, as users will be interested in "how it works," before trusting it completely. In particular, in safety critical systems (e.g., air traffic, power plants, hospitals) it is imperative to know the behavior of an application under all possible input conditions. How can users trust a machine if they do not know how it works or what its behavior will be under extreme circumstances? Neural networks learn from examples, and examples of extreme circumstances are rare by their very nature.

## 1.1 Background

In the development of application systems, there is a notable difference in system designs based on the use of artificial neural networks, and other system designs.

Typical traditional application development has evolved to contains such steps as *problem definition*, *identification of system variables*, *data collection or creation*, *making a model*, *testing the model*, *prototype construction*, *evaluation*, etc. In most application systems, three layers of abstraction can be identified (see also Figure 1.1):

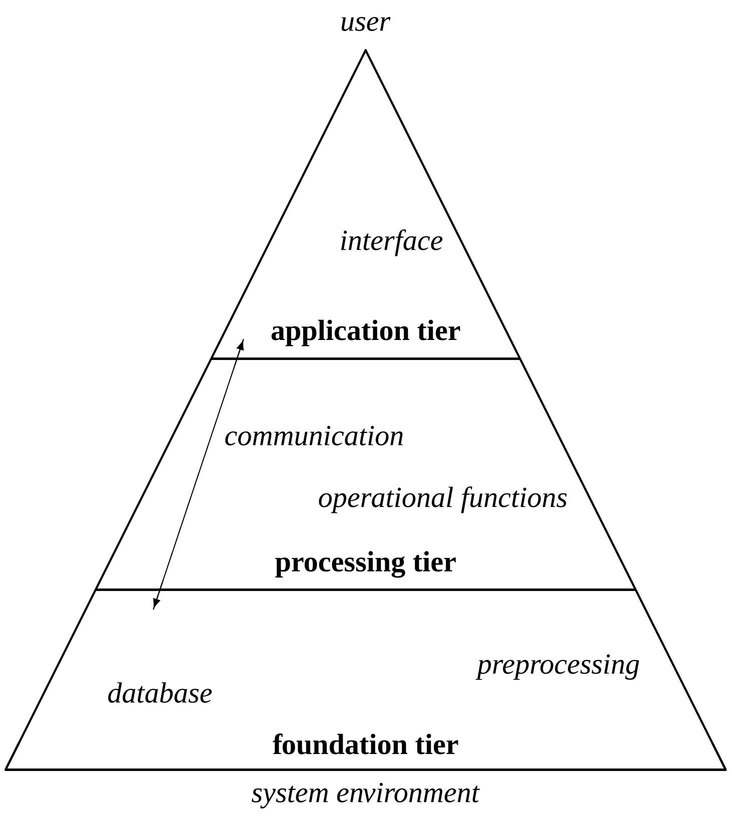


Figure 1.1: The 3-tier model.

1. a high-level layer is the system's window to the world; it contains application-specific elements such as a user interface;
2. a mid-level layer is the system's machine room; it contains operational functions, ideally in modular blocks that can easily be reused for different applications; this layer also handles the communication between the top and bottom layers;
3. a low-level layer is the system's foundation; it contains data and low-level functions for initial data processing and for modeling the physical properties of the environment in which the system is operating.

This is called a *3-tier model*, see Figure 1.1. The high-level layer or top tier is often called user tier or application tier. The mid-level layer or middle tier is also called processing tier or business logic tier (e.g., in Web-based applications). The low-level layer or bottom tier is also called foundation tier or database tier.

Whether explicitly or implicitly, the 3-tier model commonly forms the base for system designs in many applications. As long as the communication flow between the tiers or layers is well-defined, the individual tiers can be implemented independently from each other. Furthermore, if the 3-tier model is well-designed and implemented correctly, the mid-level and low-level tiers can be reused in different applications. Ideally, the bottom tier can be partially or completely used in all application systems operating in the same environment. The middle tier usually contains a subset of possible modules that can be found within the application field in which the system is used.

In contrast with many modern application systems that do not make use of neural networks, traditional neural network applications are often single-tier systems. The application design of 1-tier systems is an all-in-one design. One block, or module, takes care of all system tasks, including data processing, functional processing, and user interfacing. Whilst this may still lead to a satisfactory solution for the particular problem for which it was designed, at the same time it results in a solution that is generally unscalable, is practically useless for solving different yet related problems, and is hard to analyze.

In a single-tier system the process knowledge is distributed among all elements of the system. This makes it very difficult to extract the contained knowledge. Therefor, it also becomes hard for the user to learn from the knowledge that is contained in such a system. In a neural system, such knowledge would have been extracted from the system's environment and stored in the system during the neural network's learning phase.

Since the early development of artificial neural networks, but during the past decade in particular, researchers have tried to analyze trained neural networks in order to gain insight into their behavior (Neumann 1998). For certain applications and in certain problem domains this has been successful. In particular in decision making systems and other systems that can easily be expressed in sets of rules, major advances have been made since the development of so-called rule extraction methods (Craven and Shavlik 1994). Neural network systems with relatively few inputs can sometimes be analyzed by means of a sensitivity analysis (Hashem 1992), which is a nonparametric statistical analysis technique.

However, the data spaces in which most neural network systems operate is so high-dimensional that an extracted rule base would become too large to be easily interpreted. In addition, most neural networks are so nonlinear that a sensitivity analysis would only be valid for a small part of the input space. For these reasons, this thesis presents a generic neural network analysis method that utilizes domain-specific basic functions (Van der Zwaag et al. 2002a) that are easy to interpret by the user and that can furthermore be used to optimize neural network systems. An analysis in terms of basic functions may also make clear how to (re)construct a superior system using those basic functions. The neural network is then in fact used as a system design advisor.

The idea behind this is that it is easier to locate knowledge in a system if it is based on the 3-tier model. Basic domain knowledge is stored in the elementary functions that are found in the bottom tier. It can be easily identified or analyzed. The middle tier basically describes how this knowledge is used in the system, and the application results that are acquired on the base of the system knowledge is presented in the top tier.

In recent years modular neural networks have been developed whose design starts from the 3-tier model and uses a domain-dependent library of basic functional modules. These base modules can then be used in combination with learnable neural modules to form a heterogeneous network. Dinerstein et al. (2003) present a technique in which a single large-scale task (too complex to be performed by a single neural net) is automatically split into simpler subtasks. A multi-module system of neural nets is then trained so that one of these subtasks is performed by each net (Dinerstein et al. 2003). Whether the subtasks are obtained automatically or not, the basic functional modules have to be chosen carefully, as they can have a crucial impact on the learning behavior of the network. The modular networks created in this way require new learning strategies to ensure proper learning.

From the above, one can conclude that it would be beneficial in many aspects if systems based on a single-tier model could be transformed into equivalent systems based on the 3-tier model. This thesis proposes a generic method that can be used as a first step to achieve this for monolithic neural network systems. Whereas in general the system knowledge is stored in the neural network in a distributive manner, this thesis will show that it is possible to create a foundation tier on which the whole system is apparently based.

Starting from Chapter 4 we will present methods for breaking the internal system knowledge into identifiable basic foundation blocks. These foundation blocks, or basic functions, depend on the application domain in which the system is operational, and so do the methods to extract those basic functions. However, the domain-dependent methods are all based on a single generic domain-independent idea, namely the analysis of the neural network in terms of (generic) domain-dependent basic functions.

## 1.2 Artificial neural networks

Artificial neural networks, usually just called neural networks (NNs) (Haykin 1994, p. 1), are adaptive systems composed of (many) simple processing units operating in parallel. The network function is determined by network structure, connection weights between the processing units, and the processing performed at the processing units or computing elements.

One of the key features of neural networks is their adaptivity, their ability to learn from examples. The knowledge that they acquire from the examples, is stored in the parameters of the network. Chapter 2 will treat this in more detail.

Neural networks can be found in many varieties, depending on their implementation, the way the neurons are arranged and interconnected, the way information is processed by the network, etc. This thesis mainly deals with the most common types of neural networks with supervised learning: feedforward error-back-propagation networks (see e.g., Rumelhart et al. 1986b).

In training these networks, input information is fed to the inputs of the network, which then processes the information in parallel within layers, but sequentially between layers, hence the term “feedforward” (Rumelhart et al. 1986a). Then, the output of the network is compared to the desired output, resulting in a possible difference between the actual and the desired output patterns. This is called the output error for the particular input pattern that caused the difference. This error is then used to adjust the parameters in the network in such a way that the next time the same input is fed to the network, the error will be smaller. The network parameters are basically the weights of the connections between the neurons in the network. The error is back-propagated through the network, as a particular neuron’s influence on the error depends on that neuron’s connection to the network output, i.e., it depends on the influence of the neurons and connections that lie closer to the output layer (Werbos 1974). After training has finished, the networks use the feedforward component only. As this thesis discusses the analysis of this particular type of neural network, it will be treated in more detail in Section 2.5.

## 1.3 Analysis of neural networks

Knowledge that has been acquired by a neural network during the learning stage, is stored in the synaptic weights, the strengths of the connections between the neurons of the network. One of the major issues when dealing with neural networks, is how to accurately and efficiently extract this knowledge from the network. Knowledge extraction is useful, for example because it enables to view the neural network system from a 3-tier model point-of-view (Van der Zwaag et al. 2002c).

Because the network architecture plays an important role in the knowledge extractability, knowledge extraction can be a problem that is very difficult to overcome (Neumann 1998). This thesis treats a new idea in knowledge extraction from multilayer feedforward neural networks, which are the most widely used neural network type (Sarle 2001, Part 1), yet also one of the most difficult types of networks from which to extract the learned knowledge (Golea 1996).

## 1.4 Outline of this thesis

The remainder of this thesis is organized as follows.

Chapter 2 reintroduces artificial neural networks, and gives a brief overview of the enormous variety of neural networks and applications that have been developed since the first mathematical model of human nerve cells was described.

A review of existing methods for the analysis of neural networks is presented in Chapter 3. This includes a discussion to what extent these existing methods are successful in retrieving complete and comprehensive knowledge from the network.

In Chapter 4 a study is presented in which we identify those application domains for which existing knowledge extraction techniques produce insufficient results. This is then used as a starting point for exploring the suitability of existing and new methods for neural network analysis in those domains. This is then further expanded by outlining the new theory of the analysis of trained neural networks in terms of domain-dependent basic functions, the main topic in this thesis. Based on this theory, suggestions for basic functions are given for a range of application domains.

Two of these suggestions are then worked out in more detail in Chapters 5 and 6 and applied to some typical applications in the respective problem areas. Therefor, it is expected that those two chapters offer good illustrations of the benefits of the new method for neural network analysis presented in this thesis.

Chapter 7 concludes this thesis and briefly explores ideas for the future.

# Chapter 2

# Artificial neural networks

This chapter reintroduces artificial neural networks. Sections 2.1 and 2.2 describe the historical background of the mathematical model on which such neural networks are based. Section 2.3 gives a brief overview of the enormous variety of networks that have been developed since the first mathematical model of human nerve cells was described by McCulloch and Pitts (1943). A few main types are highlighted in Sections 2.3.1 to 2.3.3.

Some of the advantages and disadvantages of artificial neural networks are listed in Section 2.4.

This thesis discusses knowledge extraction methods for one particular type of neural network: feedforward–error-back-propagation networks. A more detailed introduction to this type of network is presented in Section 2.5.

To conclude this chapter, Section 2.6 gives an impression of the types of applications in which artificial networks have been involved. One example of an application of self-organizing maps is highlighted in Section 2.6.1.

## 2.1 The artificial neuron

As can be seen from Figures 2.1 and 2.2, a single artificial neuron, such as is also used in multilayer feedforward neural networks, is a very simplified mathematical model of a biological neuron. The basic concepts that form the inspiration for the artificial neuron, are the following:

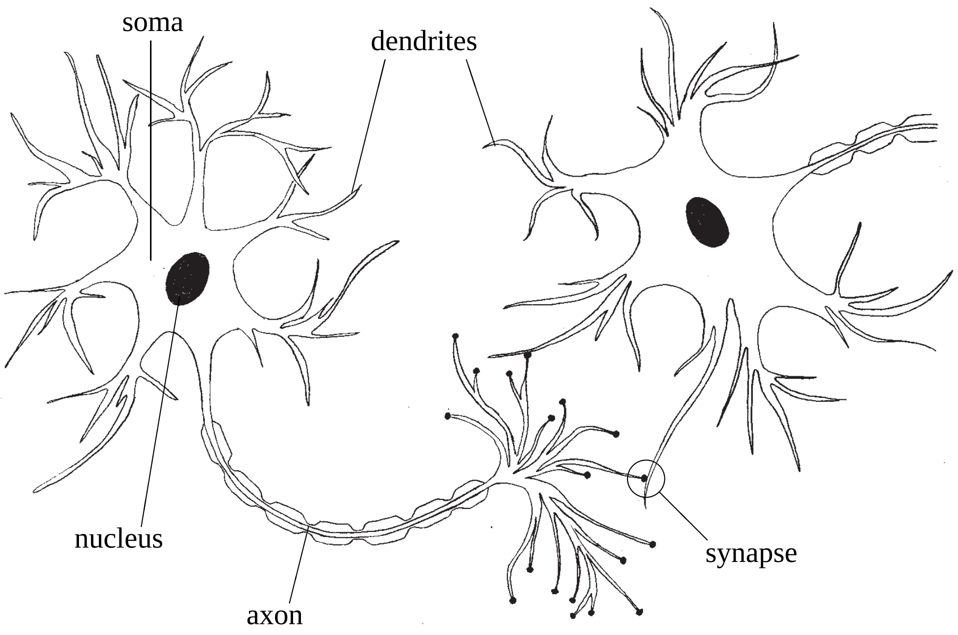


Figure 2.1: An illustration of a biological neuron.

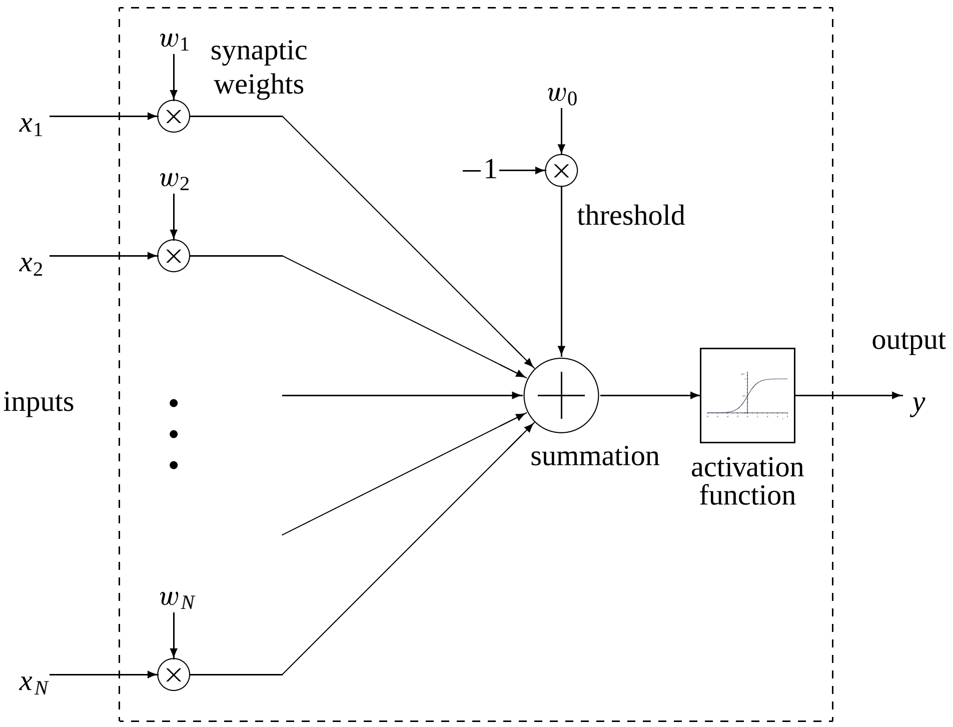


Figure 2.2: An artificial neuron is a mathematical model of a biological neuron.

**Synaptic weights** are a way to express the importance of the connections that carry the values of input signals to the neuron under investigation. The synaptic weights can be positive, negative, or zero. The corresponding inputs are excitatory, inhibitory, and inactive signals. The biological equivalents are formed by the synapses: the points where presynaptic ends of one neuron's axon meet with dendrites of other neurons. Electrical signals traveling through the axon are transferred to a dendrite belonging to another neuron by means of the release of chemicals in the axon, which are then received by the other neuron's dendrite. This in turn causes an electrical signal to travel through the dendrite towards the soma, the neuron's main body. The number of chemical particles (*transmitters*) that is transferred within a synapse is a measure for its strength (Shepherd and Koch 1990).

**Summation of weighted inputs** reduces the multiple inputs to the neuron to a single value. This single value then allows for a one-dimensional activation or transfer function, for which it is the only input. In the biological neuron, the total amount of electrical stimuli received by the soma determines whether the neuron will activate or not.

**Activation** of the neuron is controlled by the activation (or transfer) function. This takes the weighted sum of the inputs as its input and yields an output signal usually limited between a minimum and a maximum value. The activation function can be a simple threshold, below which the neuron is said to be inhibitive, and above which it is said to be excitative. The output signal for a single neuron is one-dimensional, i.e., it can only have a single value at a given moment in time. In the biological neuron, if the total amount of electrical stimuli received by the soma is higher than the neuron's activation threshold, the neuron will activate, or *fire*. This activation threshold is typically modeled in the artificial neuron by an extra weight connected to a so-called bias or threshold unit that delivers a constant output of $-1$.

In general, the activation function is a nonlinear nondecreasing function bounded between $-1$ and 1 or between 0 and 1, such as in Figure 2.3. Other examples can be seen in Section 2.5. As mentioned above, included in the weighted sum of inputs can be an internal variable, called the threshold value. This allows the activation function to shift along the input axis. The threshold value is usually implemented as an extra synaptic weight. Its synapse is then a connection to the neuron from a bias unit that has a constant output value of $-1$. The bias unit therefor has no processing ability.

The output signal of the neuron is forwarded to neurons whose inputs are connected to the output of the neuron under investigation. In case of a neuron in the output layer of the network, the output signal is an output of the neural network. Because

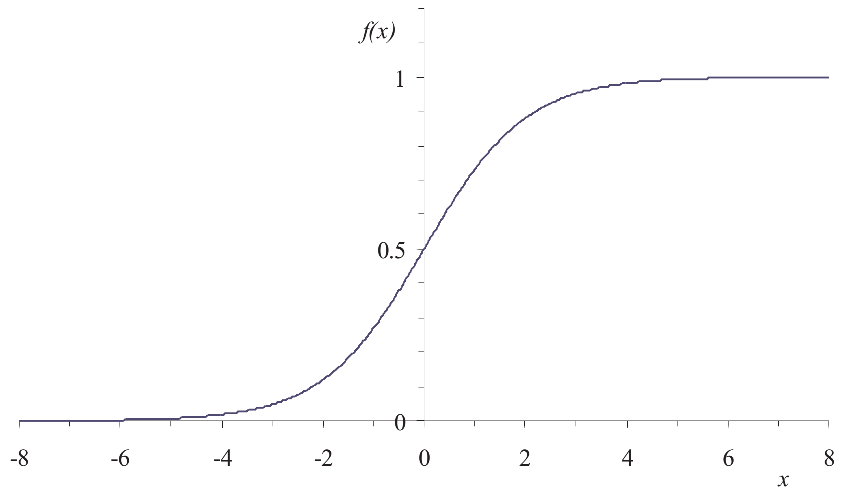


Figure 2.3: A sigmoid activation function, $f(x) = \frac{1}{1+\mathrm{e}^{-x}}$.

each neuron has exactly one 1-dimensional output, the number of output neurons is also the number of network outputs. So, suppose a network has $N$ output neurons, then it also has $N$ outputs. Or, one can say that the network output is then $N$-dimensional.

A “network” consisting of a single neuron is called a *perceptron* (Minsky and Papert 1969). Perceptrons are only capable of solving linear separation problems, such as illustrated in Figure 2.4. In linear separation problems, the input space can be divided by a plane. Input values on one side of the plane add up to a weighted sum value below the threshold, and the weighted sum of input values on the other side of the plane is above the threshold. So, depending on from which side of the plane the input values are coming, the perceptron will be in either inhibitory or excitatory activation mode, thus allowing classification of the input.

## 2.2 Networks of neurons

Artificial neural networks, usually just called neural networks (NNs) (Haykin 1994, p. 1), are adaptive systems composed of many simple processing units operating in parallel. The network function is determined by network structure, connection weights between the processing units, and the processing performed at the processing units or computing elements.

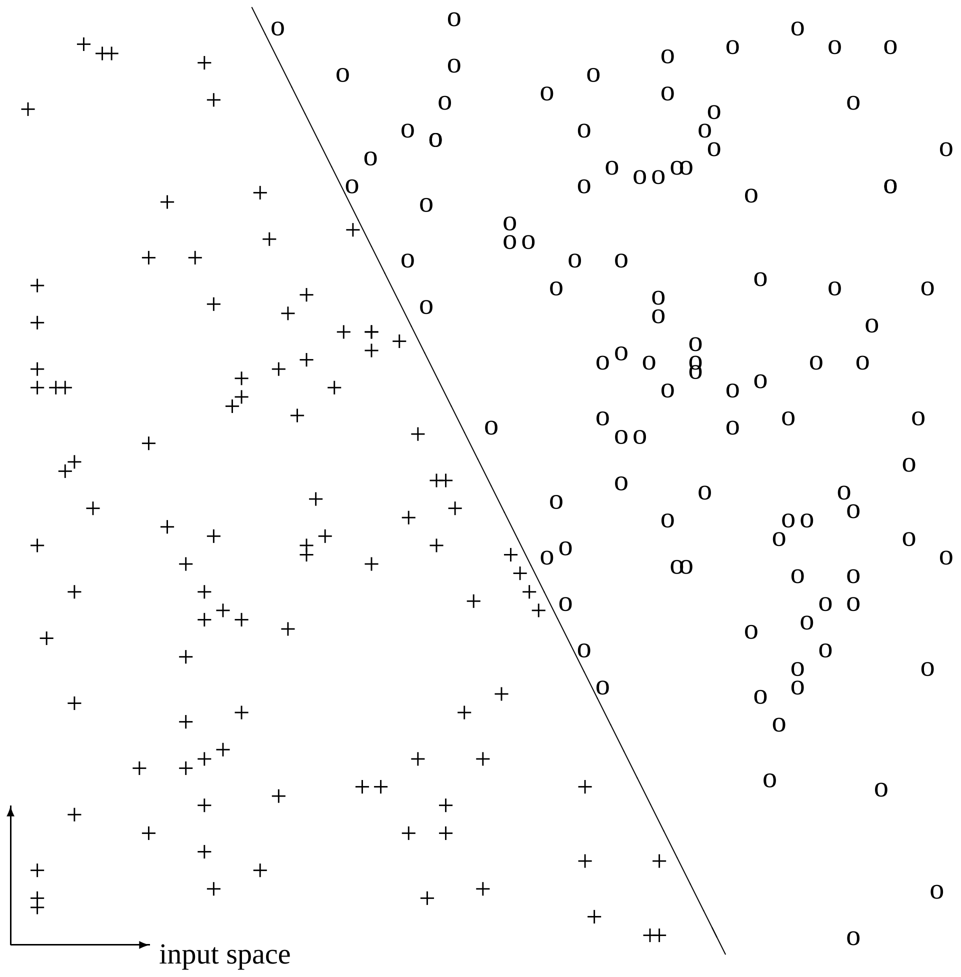


Figure 2.4: A linearly separable 2-class system. Samples of both classes ("o" and "+") are shown along with the corresponding linear class boundary.

The simple processing elements in the artificial neural network are called (artificial) neurons, analogous to the human nerve cells after which they were first modeled by McCulloch and Pitts (1943). Figures 2.1 and 2.2 (Section 2.1) show a representation of a biological neuron and an artificial neuron, respectively. The inputs to the artificial neuron are either the network-wide inputs or internal inputs. They are identical to the network inputs if the neuron is directly connected with the input layer of the network. If they are connected to outputs from other neurons, they are (internal) layer inputs. This is also the case if the inputs are connected to outputs from time delay units in computer simulations of certain types of neural networks (see Section 2.3.2 for an explanation).

Biological neurons are interconnected via synapses, which chemically transfer signals from the axon of one neuron to dendrites of other neurons (Shepherd and Koch 1990). The strengths of the synapses are modeled by the weights in the artificial neuron. The dendrites transport electric pulses ("weighted inputs" in the artificial neuron) to the processing center or soma, which, depending on the combined inputs ("sum"), may send out an electronic pulse through the axon ("output"). This so-called firing of the neuron is modeled by the artificial neuron's activation function.

Human brains are estimated to contain enormous amounts of neurons, in the order of $10^{10}$. They are connected by even more synapses, about $6 \cdot 10^{13}$ connections (Shepherd and Koch 1990). The largest amount of neurons in an artificial system sofar is "only" $3.77 \cdot 10^{7}$, according to the *Guinness Book of World Records* (Footman and Young 2001), and it was never actually built, due to financial trouble (De Garis 2002).

Human brains never stop learning, although most of the connections are formed during the first two years (Haykin 1994). Most artificial neural networks, however, have a distinct learning stage. After training has finished, the network parameters are fixed and the adaptivity feature of the neural network is no longer used. In modern artificial neural networks, more and more exceptions to this distinctive two-stage paradigm will appear, and "life-long learning" (Hamker 2001) will become more common.

Knowledge that has been acquired during the learning stage, is stored in the synaptic weights, the weights of the connections between the neurons of the network. One of the major issues in dealing with neural networks, is how to accurately and efficiently extract this knowledge from the network. Because the network architecture plays an important role in the knowledge extractability, this can be a problem that is very difficult to overcome. This thesis treats a new idea in knowledge extraction from multilayer feedforward neural networks, which are the most widely used neural network type in practical applications (Sarle 2001, Part 1), yet also one of the most difficult types of networks from which to extract the learned knowledge (Andrews et al. 1995).

## 2.3 Neural network types

Neural networks can be found in many varieties, depending on their implementation, the way the neurons are arranged and interconnected, the way information is processed by the network, etc. A rough estimate is that there are currently between 40 and 50 main types (Sarle 2001):

- the Perceptron (Minsky and Papert 1969, Rosenblatt 1958),
- Adaline and Madaline (Widrow and Hoff 1960),
- error-backpropagation nets (Werbos 1974, Rumelhart et al. 1986a, 1986b),
- Boltzmann and Cauchy machines (Takefuji and Szu 1989),
- time-delay neural networks (Waibel et al. 1989),
- Artmaps (Carpenter and Grossberg 1987),
- extended Kalman filters (Han and Veloso 1997, Kalman and Bucy 1961),
- Learning Vector Quantization (Kohonen 1986, Kohonen 1990),
- probabilistic neural networks (Specht 1988),
- simulated annealing (Kirkpatrick et al. 1983),
- self-organizing maps (Kohonen 1982),
- real-time recurrent learning (Williams and Zipser 1989),
- learning matrices (Steinbruch and Piske 1963),
- associative memories (Kosko 1988),
- differential competitive learning (Kosko 1992),
- Brain-State-in-a-Box (Anderson et al. 1977),
- shunting Grossberg (Grossberg 1973),
- discrete and continuous Hopfield networks (Hopfield 1982),
- and many more.

All of these categories can be subdivided into even more subtypes. Using other characteristics rather than architecture as the distinctive feature, different categorizations can be made. For example, neural networks can be categorized into two main groups, where the distinction lies in the learning method:

- supervised learning: the network is trained with examples of input and desired output. Networks in this category try to find a mapping from the input space onto the output space.

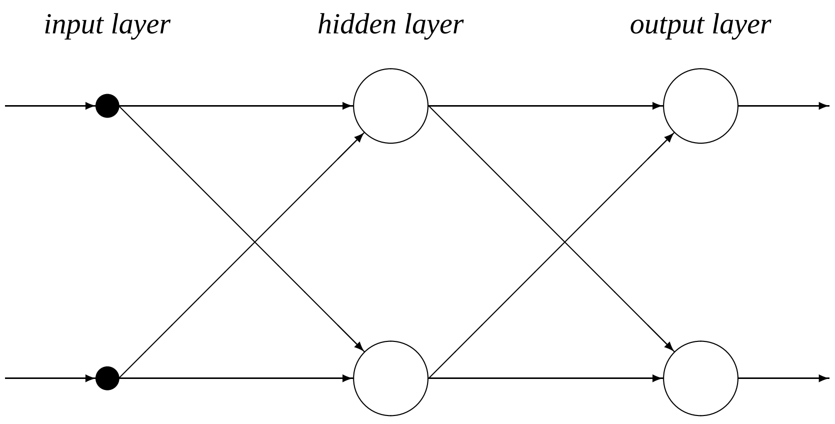


Figure 2.5: A schematic representation of a feedforward neural network, with 2 inputs, 2 neurons in a single hidden layer, and 2 outputs, where each layer is fully connected to the next layer. This is generally meant when a neural network is said to be *fully connected*.

- unsupervised learning: the network tries to organize the input data in a useful way without using external feedback. Networks in this category are often labeled after the training stage, in order to, for example, use the obtained network as a classifier (see, for example, Iordanova et al. 1992). Labeling means that an expert assigns classes to the feature vectors stored in the network.

Both of these groups can be subcategorized into strict feedforward networks and networks which contain feedback loops, which require different learning strategies. In the following subsections we will highlight some general types of neural network architectures.

### 2.3.1 Feedforward networks

The most common types of neural networks with supervised learning are feedforward error-back-propagation networks (Sarle 2001, Part 1). These are multilayered continuous-valued perceptron-like neural networks (see e.g., Rumelhart et al. 1986b). Some examples are shown in Figures 2.5 to 2.8. Feedforward networks are organized hierarchically, layer by layer, with an input layer (in which actually no processing units are present) on one side and an output layer on the other side. The layers in between are so-called hidden layers. Within each layer of neurons, the neurons are operating in parallel.

Most examples of applied neural networks found in literature are of the fully con-

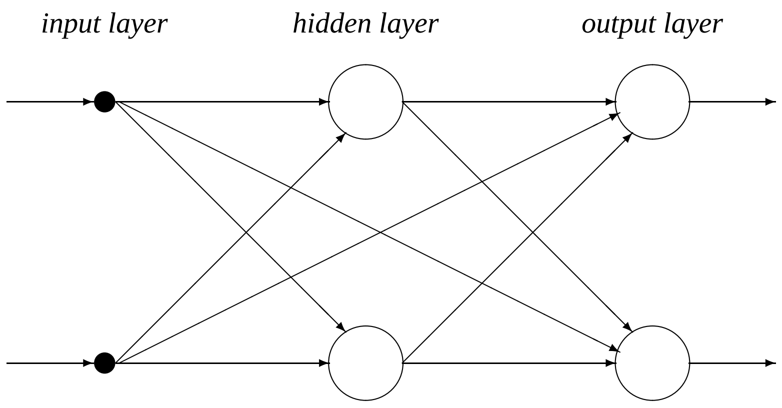


Figure 2.6: A schematic representation of a feedforward neural network, with 2 inputs, 2 neurons in a single hidden layer, and 2 outputs, where each layer is fully connected to each of the following layers. This type of full connectivity is seldom used in practice. Actually, if the network has just one hidden layer, as in the example shown here, the only difference with a network in which only consecutive layers are fully connected, is that the inputs are forwarded to all processing layers, rather than just to the first hidden layer.

nected type (Elizondo and Fiesler 1997) seen in Figure 2.5. A neural network is said to be *fully connected* if, for every layer, each node in the layer is connected to each node in the subsequent layer in the network hierarchy. Fully connected neural networks that additionally have synaptic connections between layers that are not adjacent to each other, such as shown in Figure 2.6, are rarely encountered in literature. Because most error-back-propagation training methods are based on a layer-to-layer architecture (Rumelhart et al. 1986b), different learning techniques are required for such networks. For the 2-neuron example shown in Figure 2.7 it was quite straightforward to deduce (in a Boolean logical way) values for the synaptic weights in order to let the network perform the “exclusive or” (XOR) function.

Partially connected neural networks such as seen in Figure 2.8 are occasionally found in literature (Elizondo and Fiesler 1997). A neural network is said to be *partially connected* if, for one or more layers, one or more nodes in the layer are not connected to one or more nodes in the subsequent layer in the network hierarchy. Some synapses are missing, in comparison with the fully connected neural network of Figure 2.5. Partially connected neural networks are in particular used in systems applied to time series (Haykin 1994). Often, these neural networks are *locally connected*, which means that local groups of synaptic connections can be identified in the network. For example, the hidden neurons may each be connected to a neighborhood subset of inputs, in which case it is said that these neighboring

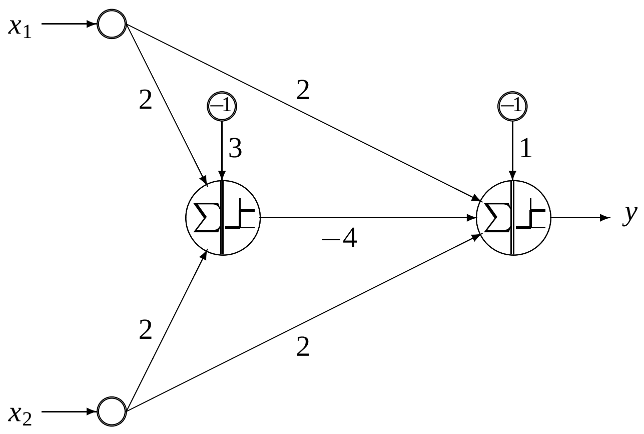


Figure 2.7: A feedforward neural network with two inputs, one hidden neuron, and one output neuron. Synaptic weights are shown along the connections between the units. Note that the inputs are also directly connected to the output neuron. Using threshold activation functions, this network performs the logical XOR operation on the Boolean inputs: $y = 0$ if $x_1 = x_2$ and $y = 1$ if $x_1 \neq x_2$.

inputs form the *receptive field* of the hidden neuron to which they feed input information. This can be useful when neighboring inputs (e.g., in a time series or in an image pattern) are correlated. Information fed to the network is first processed in local subsets of neurons, and then gradually spreads through larger subsets of subsequent layers. Locally connected neural networks presume a priori knowledge about characteristics of the input patterns offered to the network. In this sense, one might regard local connectivity as a form of modularity, but, nonetheless, locally connected neural networks are still monolithic, be it with structural restrictions.

During the training or learning stage of a feedforward neural network, input patterns are fed to the inputs of the network, which then processes the information until a result is obtained from the output layer. Within each layer, the information received from the layer above (in a layout where the inputs are on top and the outputs are at the bottom) is processed by the layer's processing units, or neurons, in parallel. (In practice, neural networks are often simulated on a computer, so in single-processor computer simulations it will not be possible to perform really parallel processing within each layer of the simulated neural network.) As the processing in each layer depends on the results of the processing in the layer above, information travels sequentially between layers, hence the term "feedforward." Then, the output of the network is compared to the desired output pattern, resulting in a possible difference between the actual and the desired output patterns. This is called the output error for the particular input pattern that caused the

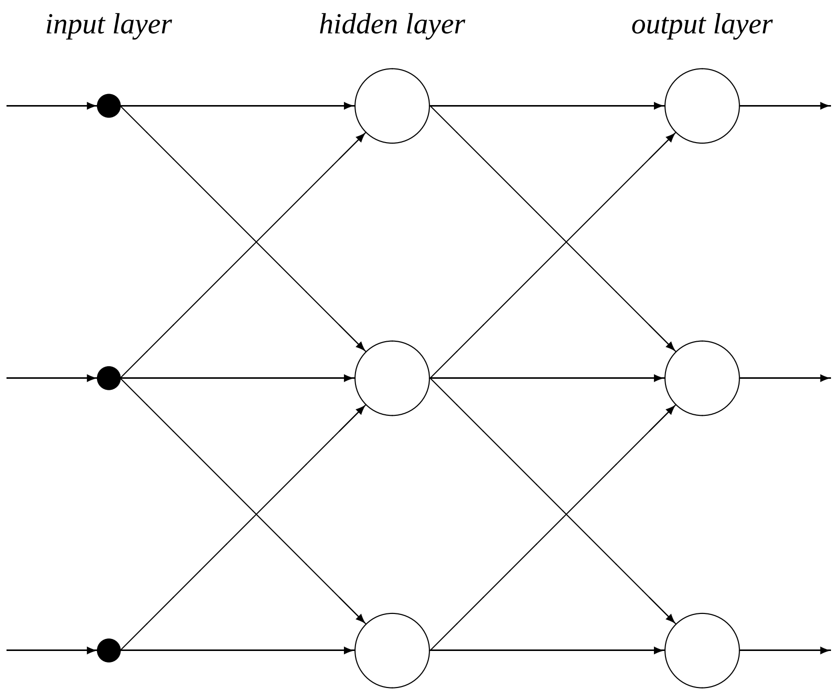


Figure 2.8: A schematic representation of a feedforward neural network, with 3 inputs, 3 neurons in a single hidden layer, and 3 outputs. A neural network is said to be *partially connected*, if 1 or more nodes in one layer are not connected to 1 or more nodes in the next layer, such as shown here. Note that this network is also *locally connected*, as subsets of neighboring synaptic connections can be identified in the network.

difference. This error is then used to adjust the parameters in the network in such a way that the next time the same input pattern is fed to the network, the error will be smaller (Rumelhart et al. 1986a).

The network parameters are essentially the weights of the connections between the neurons in the network. The error is back-propagated through the network, as a particular neuron's influence on the error depends on that neuron's connection to the network output, i.e., it depends on the influence of the neurons and connections that lie closer to the output layer. This accounts for the term "error-back-propagation." After training has finished, the networks use the feedforward component only. As this thesis mainly discusses the analysis of this particular type of neural network, it will be treated in more detail in Section 2.5.

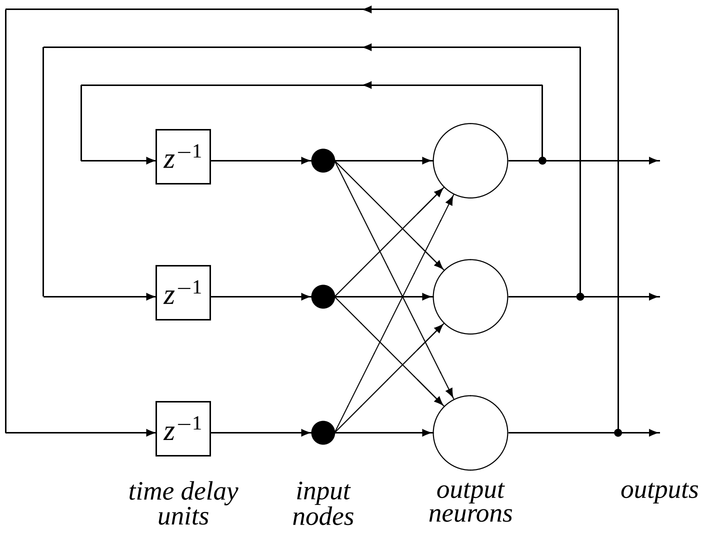


Figure 2.9: Schematic representation of a recurrent network. This fully connected example has 3 inputs and 3 outputs. This particular example does not have any hidden neurons. Output signals from the output side of all 3 neurons are fed back to each other's as well as their own input sides through the time delay operators. Once an input pattern is offered to the input nodes, the signals start to loop through the network until a stable state is reached. After that, the output can be read from the output neurons.

### 2.3.2 Recurrent networks

Feedback networks or recurrent networks, such as shown in Figures 2.9 and 2.10, are increasingly used (Medsker and Jain 2000). These exist in a range of varieties (Hopfield networks (Hopfield 1982), for example), but have in common that input into a neuron can originate from neurons both higher and lower in the network hierarchy. In other words, these types of neural networks contain loops, such as shown schematically in the example of Figure 2.9. Many recurrent networks do not have explicit outputs such as appear in feedforward networks. Instead, an input pattern offered to the network results in dynamic nonlinear behavior, due to the feedback loops that are present in the network. When the network has reached a stable state, the network state can be read and serves as output pattern (Hopfield 1982). Nevertheless, recurrent networks do exist (see, e.g., Wermter 2000) which have explicit input, hidden and output neurons. A schematic example of such a network is shown in Figure 2.10.

In computer simulations of recurrent networks, time delays (or time delay units) have to be explicitly defined in order to enable feedback processing. For this rea-

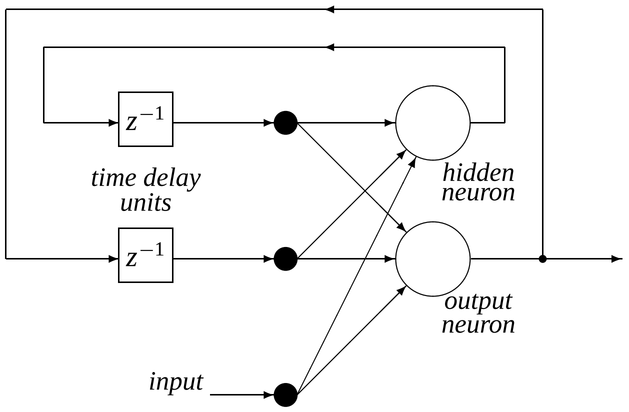


Figure 2.10: Schematic representation of a recurrent network. This example has 1 input and 1 output. In addition it has one hidden neuron. Output signals from the output side of both (hidden and output) neurons are fed back to each other's as well as their own input sides through the time delay operators. Once an input signal is offered to the input node, signals start to loop through the network until a stable state is reached. After that, the output can be read from the output neuron.

son the term *time-delay networks* is also used for this type of networks (Waibel et al. 1989). In Figures 2.9 and 2.10 the time delay units are denoted by $z^{-1}$. Furthermore, in single-processor computer simulations it will not be possible to perform really parallel processing within each layer of the simulated neural network, as was already the case in feedforward networks.

### 2.3.3 Lattice networks

A range of artificial neural networks developed by Kohonen (1982, 1986, 1990) are among the most commonly used networks with unsupervised learning (Sarle 2001), though he also developed some supervised neural algorithms, such as Learning Vector Quantization (Kohonen 1988). His self-organizing feature maps (Kohonen 1982) are examples of so-called lattice networks. In this type of neural network, the layout of the network is in the form of a lattice or map, as schematically shown in Figure 2.11. In the training stage, the neural net reorganizes the input space in such a way that similar input vectors will be represented close to each other in the map and more distinct vectors further apart from each other.

As already mentioned, the self-organizing map consists of a number of neurons laid out in a grid or lattice. This grid is often rectangular (Kohonen 1995), but other arrangements are also used. Examples are one-dimensional lattices (Ritter

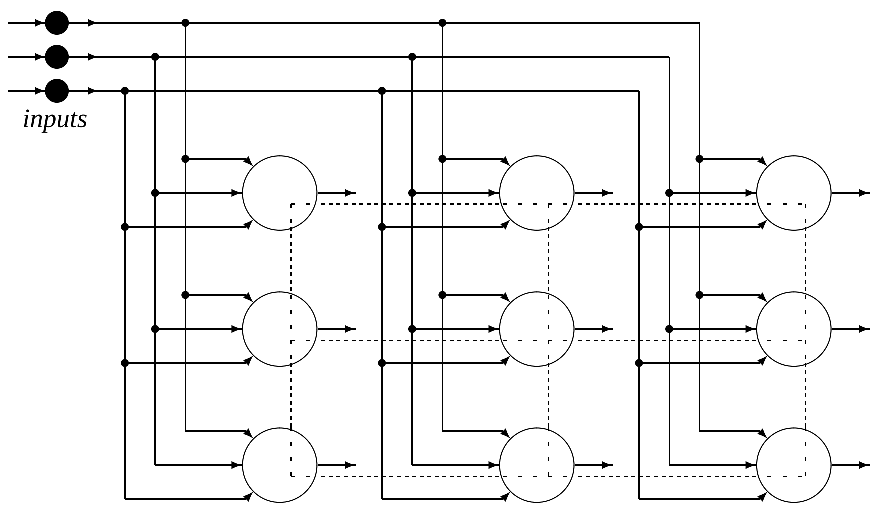


Figure 2.11: Schematic representation of a lattice network with 9 neurons on a $3 \times 3$ rectangular lattice (shown here as broken lines). In this example there are 3 inputs, so the neurons represent feature vectors containing 3 elements per neuron. The 3 inputs are fed to all neurons in the network simultaneously.

et al. 1992) and hexagonal lattices (Georgakis et al. 2001). Each neuron contains (essentially *is*) a randomly initialized feature vector with just as many elements as the number of elements in the input patterns. The lateral distance between two neurons is the shortest distance between them in the map, usually defined along a straight connecting line or along the grid lines. Depending on the lateral distance between two neurons, they can be either inside or outside each other's neighborhood. The neighborhoods are important for the adjustment of the feature vectors during learning.

During the learning or self-organizing stage, training input patterns are offered to the network one by one. For every neuron, the difference (usually Euclidian distance) between its feature vector and the input vector is calculated. The neuron with the smallest difference is then said to be the winning neuron for that particular input example.

The winning neuron's feature vector is then adjusted in such a way that the difference with the input example decreases. This is also done for the neurons in the winning neuron's neighborhood, though the larger the lateral distance from the winning neuron, the smaller the adjustment. Often the neurons immediately outside the winning neuron's neighborhood are adjusted in the opposite direction, in order to increase the vector difference between the neurons (Kohonen 1995). In that way, the network is spreading out to cover the complete input space as much

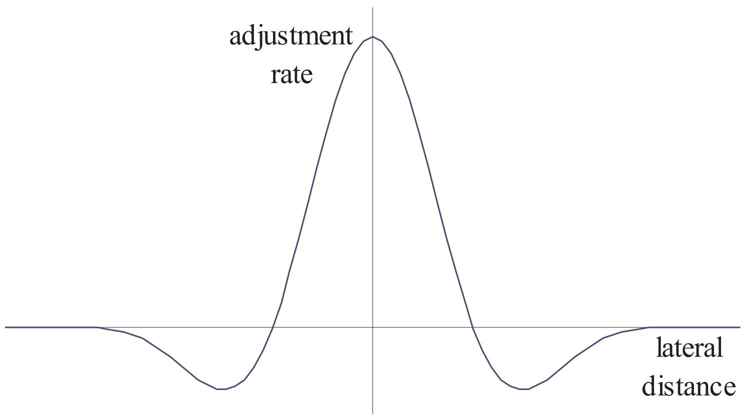


Figure 2.12: Illustration of a Mexican hat function.

as possible. The rate of adjustment as a function of the lateral distance to the winning neuron usually has the shape of a Mexican hat, such as the function shown in Figure 2.12. Discrete versions (Haykin 1994) are in principal equivalent, since the network lattice is also discrete. Over time, this hat is shrinking both horizontally (the neighborhood is getting smaller) and vertically (the adjustments are getting smaller), until the learning stage is over.

An example of a self-organized feature map can be seen in Figure 2.13. In this figure, the neurons are represented by the feature vectors that are contained in them. The result shows prototypes of fragments of handwritten zeros, organized in a 2-dimensional lattice. This $7 \times 7$-neuron self-organizing map is part of a larger system that has been trained to recognize handwritten digits (Van der Zwaag 2001), see also Section 2.6.1.

## 2.4 Strengths and weaknesses of neural networks

### 2.4.1 Strengths

Neural networks distinguish themselves from other systems by a number of useful characteristics, some of which are listed here.

First, neural networks use relatively straightforward methods of acquiring knowledge through learning from examples. They can do this even from very high-dimensional data (Haykin 1994). Basically, they map the input space to the output space, without being told explicitly how to accomplish this mapping.

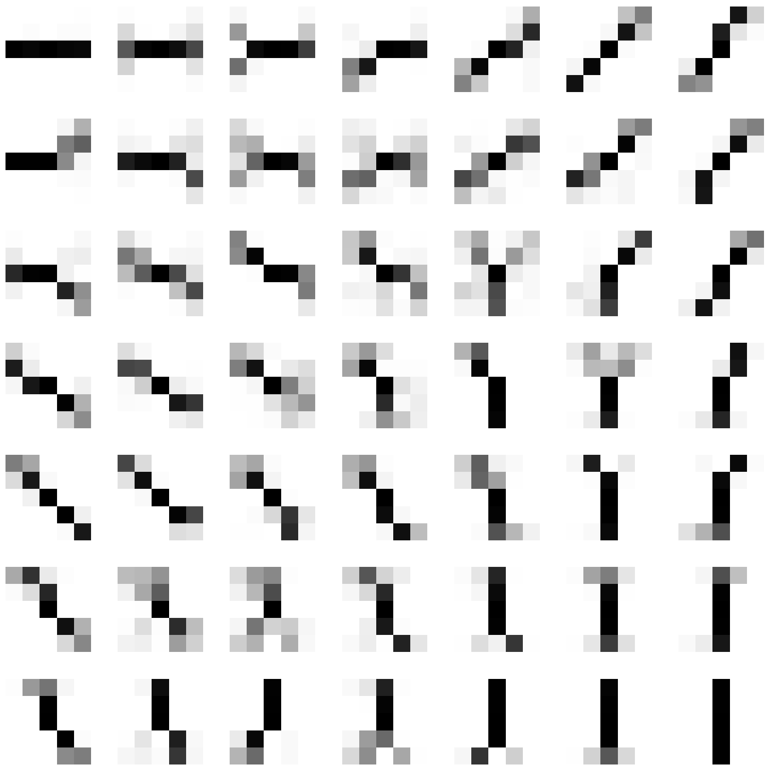


Figure 2.13: Result of self-organization of fragments of handwritten digits (zeros only).

Second, neural networks store large amounts of acquired information in a very compact form. In this sense, a neural network is a representation of data, stored in an intelligent knowledge framework. In this way, it can also be used as a memory (Ruppin and Yeshurun 1991), or as a tool for data compression (Verma et al. 1997).

Third, neural networks are able to generalize, i.e., high degrees of accuracy have been reported when trained neural nets are applied to previously unseen examples (e.g., Rosario and Hearst 2001). A neural network can make an implicit model of the environment in which it is operating, thus eliminating the need for explicitly formulating such a model, which is often difficult or expensive to achieve.

Fourth, neural networks can tackle problems that are strongly nonlinear. This is inherent to the nonlinear behavior of the individual neurons within the network. The interconnectivity between the neurons further enhances the nonlinearity of the neural network, by distributing the nonlinearity over the whole network. Nonlinearity is a highly important property, particularly if the underlying physical mechanism responsible for the generation of an input signal (e.g., speech) is inherently nonlinear (Haykin 1994).

And last but not least, neural networks are robust with respect to noise in the input data, or, for example in the case of neural network hardware, with respect to faulty connections or malfunctioning neurons (Bolt 1991). This *fault tolerance* is a very distinct feature of neural networks, and also clearly distinguishes neural networks from traditional digital computers, which are highly fault-intolerant (Hammerstrom 2000).

### 2.4.2 Weaknesses

One of the strengths of neural networks, their nonlinearity, causes at the same time one of their most significant weaknesses: the inability to satisfactorily explain their behavior. This is only true for certain neural network models, but nevertheless it is one of the major obstacles when it comes to integrate neural networks into real-world applications. This is in particular true for safety-critical systems, in areas such as medical systems, industrial processing and control, power plants, and defence (Kurd and Kelly 2003). In those areas, the role of neural systems is usually limited to that of a diagnosis advisor (Kim and Bartlett 1996, Schmitt et al. 2002). In this thesis, a method is developed to help extract the knowledge from the network in such a way that it is complete and comprehensive.

Another weakness of neural networks is the time involved in learning tasks. This of course largely depends on the task that the network is learning, on the implementation, i.e., whether the neural network is implemented in hardware or simulated in software, and on the learning algorithm. Major advances in learning algorithms have been achieved in the past years (Jain and Kacprzyk 2002), and the ever increasing speed of computers is allowing more complicated tasks to be learned in shorter time.

Tuning learning and architectural parameters of a neural network can be a hideous task, but here too advances are made (e.g., LeCun et al. 1993). In particular in hybrid systems, e.g., neural systems combined with fuzzy or evolutionary systems, often the architectural design or the tuning of parameters is taken over by other computational intelligence methods (e.g., Vonk et al. 1995).

Finally, a very common weakness of applied neural networks is the lack of expertise of the user with respect to either artificial neural networks or the application domain. Too often, the full capabilities of neural systems are overlooked by the user. One reason for this is the fact that neural network experts are often not familiar enough with the system in which a neural network is to be embedded. On the other side, the experts in the embedding system are usually not familiar enough with neural networks. This would not be so much of a problem if there were a close cooperation between the two parties, but in practice, this is rarely the case, particularly after the embedding is finished. Adjustments or updates are often not implemented because of a lack of neural system knowledge.

An example of successful knowledge transfer between neural network experts and experts from the application domain is formed by the EUREGIO Neuro-Fuzzy Centre (NFC), later the EUREGIO Computational Intelligence Centre (ECIC) (NIWI 2001), projects that were funded by the European Commission. The ECIC brought together knowledge and experience of Computational Intelligence of the University of Twente, the Netherlands, and the Fachhochschule Münster, Germany. The main goal of this project was the knowledge transfer from research institutes to small and medium-sized enterprizes. This goal was reached by development of innovative products, carrying out feasibility studies, giving advice to small and medium-sized enterprizes, and initiation and organization of international projects (NIWI 2001). The project was presented as "NeuroFuzzyRoute in the EUREGIO" at the world exposition EXPO2000 in Hannover, Germany.

The main topic of this thesis addresses exactly the problem of how to explain the neural network to the end-user, by exploring ways to translate a trained neural network into a system that can be fully understood by the expert in the application field in which the neural network was embedded. If users understand the system they are working with, they may find more familiar ways to implement adjustments to the system if needed.

## 2.5 Feedforward–error-back-propagation networks

Section 2.3.1 already introduced feedforward neural networks, but since this thesis addresses the knowledge extraction from this type of supervised neural networks in particular, we will treat them here in more detail.

Feedforward–error-back-propagation neural networks (Rumelhart et al. 1986b) are the most commonly used neural networks that are trained with supervised learning methods (Sarle 2001, Part 1). They consist of several layers of neurons, as illustrated in Figure 2.14. Actually, the first layer, or input layer, does not contain any neurons, but merely distributes the input signals to the first layer that does consist of neurons, the first hidden layer. The output signals from the first hidden layer are then forwarded to the next layer, and so on. The final hidden layer's output signals are passed on to the output layer, which, after processing its input signals, produces the output of the network.

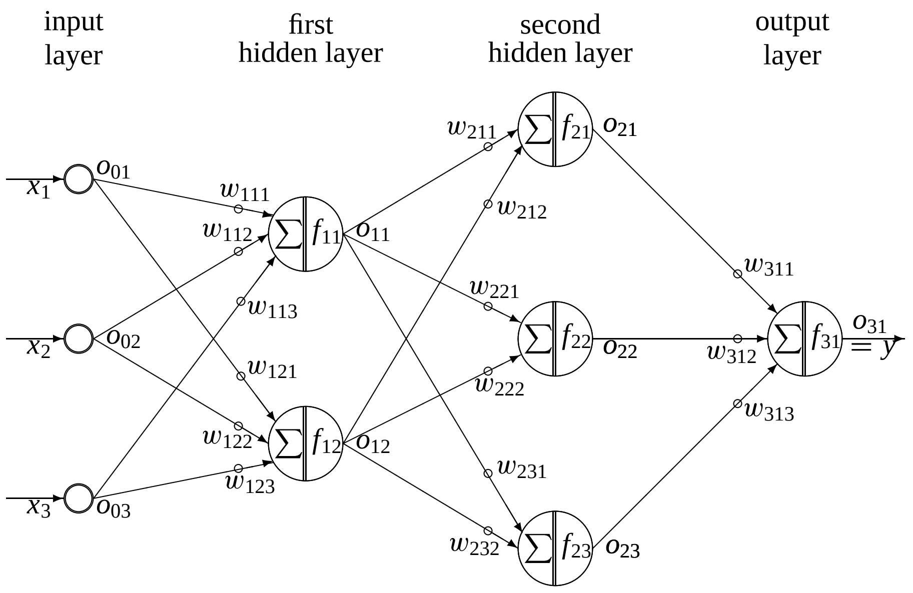


Figure 2.14: Example of a fully layer-to-layer-connected feedforward neural network with three inputs, two hidden layers with two and three neurons respectively, and one output. The threshold unit and its connections to all neurons are not shown. For clarity, the superscript $P$ in the signal names representing the pattern offered to the network is not shown either.

A feedforward neural network such as the network shown in Figure 2.14 consists of the following layers:

**the input layer,** layer 0, consisting of $N_0$ inputs $x_i^P$, with $i = 1, ..., N_0$ and $P$ being the pattern that is currently offered to the network. The input signals are then distributed as $o_0 i^P$ from the input layer to the next layer,

**the first hidden layer,** layer 1, consisting of $N_1$ hidden neurons. The outputs $o_{1i}^P$, $i = 1, ..., N_1$, of these neurons are connected to the inputs of the next layer,

•

•

•

**the *k*th hidden layer,** layer $k$, consisting of $N_k$ hidden neurons, $k = 1, ..., K-1$, where $K$ is the number of neuron layers. In a fully layer-to-layer-connected network, each neuron in this layer has $N_{k-1}$ inputs, $k = 1, ..., K-1$. The input signals $o_{k-1,j}^P$, $j = 1, ..., N_{k-1}$ to this layer are distributed along the connections from the previous layer; input signal $o^P_{k-1,j}$, $j = 1, ..., N_{k-1}$ traveling along the connection between unit $j$ in the previous layer and unit $i$ in the present layer is multiplied by the synaptic weight $w_{kij}$, $k=1, ..., K-1$, $i=1, ..., N_k$, $j=1, ..., N_{k-1}$. The weighted sum of input to neuron $i$ is then processed by the neuron's activation function $f_{ki}$ and the resulting output signal $o^P_{ki}$ of this neuron, along with the output signals from the other neurons in this layer, is forwarded to the next layer,

.

.

.

**the last hidden layer,** layer $K-1$, consisting of $N_{K-1}$ hidden neurons. The output signals $o^P_{K-1,i}$, $i = 1, ..., N_{K-1}$, from this layer are forwarded to the final layer,

**the output layer,** layer $K$, consisting of $N_K$ output neurons. The output signals $o^P_{Ki}$, $i = 1, ..., N_K$, from this layer are also the outputs $y^P_i$, of the whole network, when pattern $P$ is offered to the network.

Every layer $k$, except the output layer, furthermore contains a threshold unit, which has a constant output $o^P_{k0} = -1$ for $k=0, ..., K-1$, that is connected via connection weights $w_{k+1,i,0}$, $k=0, ..., K-1$, to neurons $i$ in the subsequent layer $k+1$.

Thus, including the thresholds, a hidden neuron $i$ in layer $k$ has a total of $n_{k-1}+1$ inputs $o^P_{k-1,j}$, a total of $n_{k-1}+1$ weights $w_{kij}$, and one output $o^P_{ki}$. An output neuron $i$ in layer $K$ has a total of $n_{K-1}+1$ inputs $o^P_{K-1,j}$, a total of $n_{K1}+1$ weights $w_{K-1,i,j}$, and one output $o^P_{K-1,i} = y^P_i$, which is also one of the outputs of the network as a whole.

The output of a neuron $i$ in layer $k$ is calculated with an activation function $f_{ki}$. Often, the same activation function is used for all neurons in the entire network, or at least in all neurons within the same layer, but this is not a requirement. Its shape usually varies from threshold functions (Figure 2.15) and piecewise linear functions (Figure 2.16) to smooth nonlinear functions such as the arctan function shown in Figure 2.17. The sigmoid function shown earlier in Figure 2.3 is the most frequently used activation function.

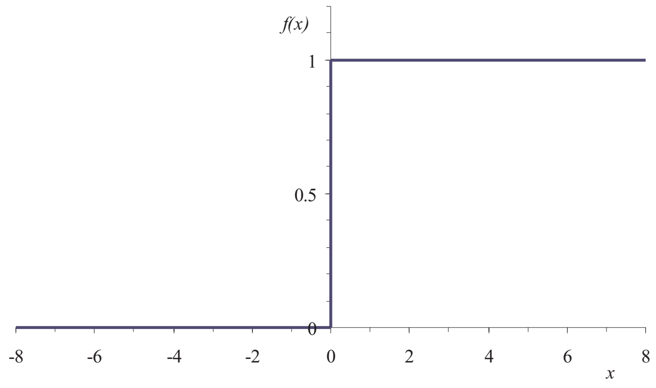


Figure 2.15: A threshold activation function, bounded between 0 and 1.

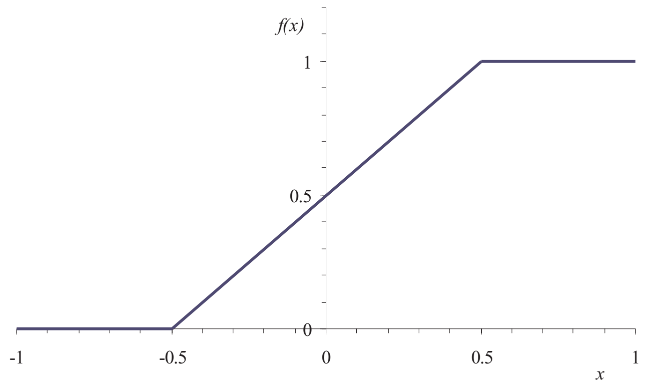


Figure 2.16: A piecewise linear activation function, bounded between 0 and 1.

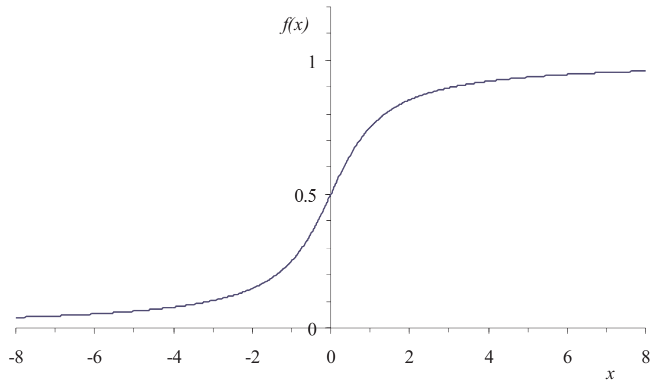


Figure 2.17: A smooth nonlinear activation function, $f = \frac{1}{2} + \frac{1}{\pi}\arctan x$.

## 2.6 Applications

Artificial neural networks have been successfully applied in almost every field one could think of (Sarle 2001), including, but not limited to, agriculture (e.g., Tao et al. 1995), finance (e.g., Trippi and Turban 1993), imaging (e.g., Muhamad and Deravi 1994), linguistics (e.g., Schmid 1994), medicine (e.g., Zhang et al. 1994), music (e.g., Griffith and Todd 1999), robotics (e.g., Miller 1994), sports (e.g., Kohle and Schonbauer 1989), telecommunication (e.g., Fritsch et al. 1993), and weather forecasting (e.g., Navone and Ceccatto 1994).

The functionality of neural networks lies in, e.g., classification, control, system identification, memory, prediction, pattern recognition, modeling, signal processing and function approximation. The most notable aspects that practically all applications of neural networks have in common is that they deal well with vague or incomplete information and that they are capable of tackling non-analytical problems using heuristic methods.

### 2.6.1 An example

An example of a neural network application is a neural network that was used for handwritten digit recognition as an educational demonstration project (Van der Zwaag 2001). Part of it was already shown in Section 2.3.3, where a self-organized feature map was presented in Figure 2.13, showing prototypes of fragments of handwritten zeros, organized in a 2-dimensional lattice. That $7 \times 7$-neuron self-organizing map was part of a larger system that was trained to recognize handwritten digits (Van der Zwaag 2001). The self-organized feature map shown in Figure 2.13 was trained to recognize parts of handwritten zeros. This map combined with nine other maps for the remaining digits formed the base for one more self-organizing map that was able to classify handwritten digits based on the subclassification of fragments of those digits.

The design of the complete system is a hybrid form incorporating both hierarchical and modular structures. A 3-tier model forms the base for the overall structure of the handwritten digit recognizing system. In this model there is a hierarchy in the level of abstraction of the different tiers. Within each tier, the structure is modular, with separate modules containing separate chunks of the knowledge that is eventually present in the system. The bottom tier contains modules for converting the handwritten digits into their electronic representations. This layer also contains modules for the preprocessing of the digits, based on a priori knowledge of handwritten digits in general. This includes operational modules for resizing, re-sampling, rotation, thinning, etc.

The middle layer contains the neural network modules. During the learning phase, there is one separate self-organizing feature map for every digit class. After the self-organization has completed, the subnetworks are combined into a larger network. This larger network contains knowledge referring to the possible shapes of digit fragments for different digit classes. A different module, again a self-organizing neural network, compiles the information from these combined subnetworks and compiles digit fragments into digit classes. This operation depends on the shape of the fragments as well as their relative prevalence in a particular input pattern.

The top layer is the user interface. It contains a module to retrieve the handwritten digit from the user, along with a module to present the result of the classification to the user.

The goal of this handwritten digit recognizer was to have a reasonably well working neural network system for demonstration purposes. Only a small number of experiments were conducted to enable comparison between various configurations from which one was to be chosen for the final demonstration setup. For training, a set of handwritten digits with 15 digits per class was used, see Figure 2.18. Testing was done with another set of 10($\times$10) examples, also shown in Figure 2.18. Previous research (Wentink (1995) and others) had yielded a maximum 92% correct classification score (percentage correctly recognized digits) on the same training and test sets. Using digit parts and using more effective preprocessing, this number now increased to 98%, which was considered sufficient for use in a demonstration.

A continuous presentation was installed at the Da Vinci Techno Activity Center in Enschede, the Netherlands, where visiting children and adults could learn about technology. The presentation consisted of a slide show explaining the whole Kohonen system of learning and recognizing, and a writing pad with display where visitors could write a digit and see the neural classifier outcome on a large screen. In the slide show, a set of unclassified digit fragments was shown to give people also a chance to determine what digit the fragments came from. The whole presentation was part of "NeuroFuzzyRoute in the EUREGIO," an exposition in the framework of the world exposition EXPO2000 in Hannover, Germany.

Considering an investigation into the knowledge that is stored in this handwritten digit recognizing neural network system, two observations can be made. The first observation is regarding the 3-tier structure of this system and its modularity. Specific knowledge is located in specific parts of the system, as every module has been designed or trained to perform specific subtasks. The second observation is regarding the type of neural networks used in this system. Self-organizing feature maps store their knowledge in the same format as the input patterns. This means

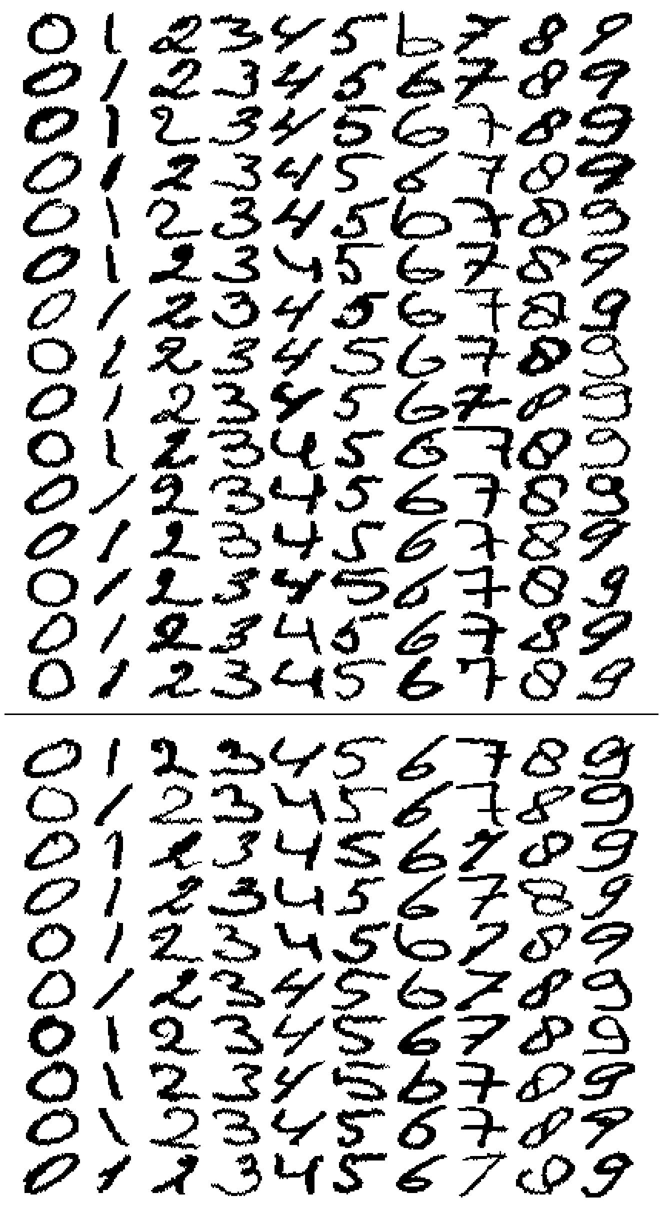


Figure 2.18: Set of handwritten digits used for training (*top*) and testing (*bottom*).

that if one is interested to see the knowledge that is stored in a network module that was trained with digit fragments of a particular digit class, the only thing needed to do is to read the feature vectors that are contained in that network. For visualization purposes, it is recommended to represent the feature vectors as colors or grey scales for easy interpretation. For example, in Figure 2.13 (Section 2.3.3) the

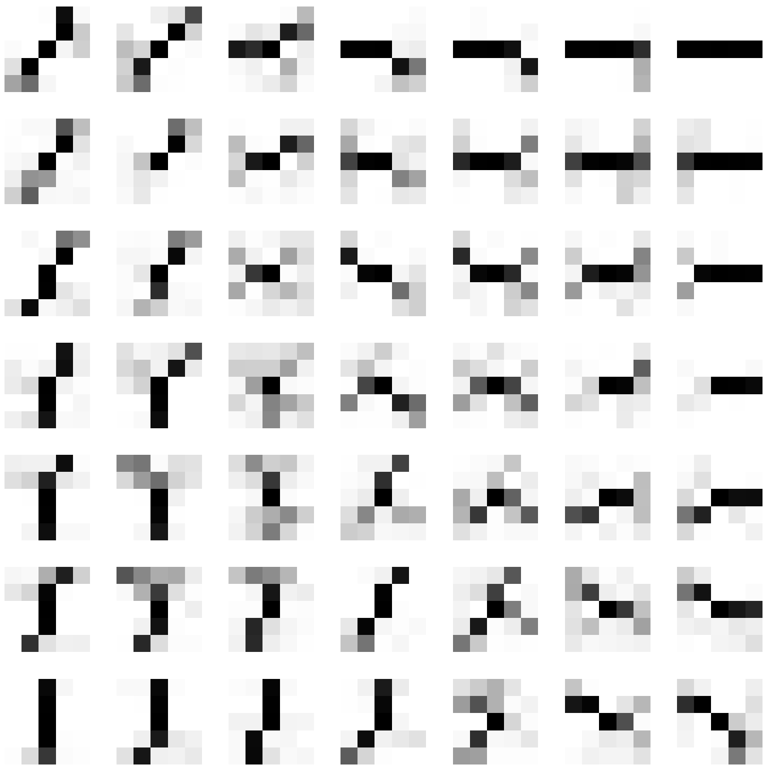


Figure 2.19: Result of self-organization of fragments of handwritten sevens.

organized fragments of handwritten zeros were shown graphically. These can easily be compared with the organized fragments of other digit classes, such as the fragments of handwritten sevens shown in Figure 2.19. One of the things that, for example, can be recognized in this figure, is the presence of fragments that show traces of intersections. These result from the training with sevens that have an extra horizontal bar (i.e., 7 instead of 7).

## 2.7 Conclusion

This chapter presented an overview of some of the most commonly used neural network architectures. In the following chapters an investigation is made into the problems that arise if one wishes to extract the learned knowledge from feedforward-type neural networks, rather than from self-organizing maps such as the ones mentioned in the previous section. We will use two approaches in this investigation.

First, an overview is presented in Chapter 3, listing some of the existing methods for the extraction of knowledge from feedforward neural networks. Furthermore, we show to what extent these existing methods are successful in retrieving complete and comprehensive knowledge from the network.

Second, a study is made to identify a selection of application domains for which existing knowledge extraction techniques produce insufficient results. This can be found in Chapter 4. Selected alternative techniques are suggested. Additionally, two case studies can be found in Chapters 5 and 6.

# Chapter 3

# Review of existing methods for the analysis of neural networks

Despite their success story, neural networks have one major disadvantage compared to other techniques: the inability to explain comprehensively how a trained neural network reaches its output; neural networks are not only (incorrectly) seen as a "magic tool" but possibly even more as a mysterious "black box" (Sjoberg et al. 1995). This is an important aspect of the functionality of any technology, as users will be interested in "how it works" before trusting it completely. In particular, in safety critical systems (e.g., airlines, power stations, hospitals) (Kurd and Kelly 2003, ERA Technology 1997, Sharkey et al. 1995) it is imperative to know or to be able to predict the behavior of an application under all possible input conditions.

The merits of the analysis of neural networks include (Andrews et al. 1995):

- provision of a "user explanation" capability;
- extension of neural network systems to "safety-critical" application domains;
- software verification and debugging of neural network components in software systems;
- improving the generalization of neural network solutions;
- data exploration and the induction of scientific theories;
- knowledge acquisition for symbolic AI systems.

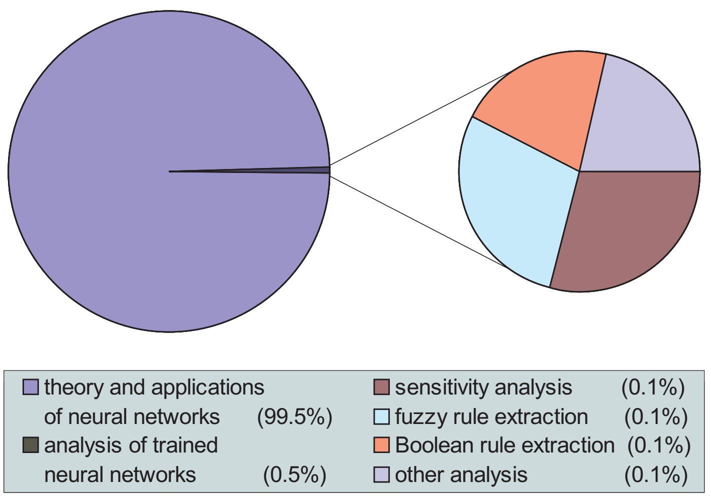


Figure 3.1: Results of a search in several major patent databases, illustrating the relatively small amount of literature on the analysis of neural networks. See also Appendix A at the end of this thesis.

These merits clearly indicate how users can benefit from the analysis of neural networks. They will see improved functionality, and by using the results of the analysis they can determine which parts of their application system are to be improved in order to get a better solution for their particular problems.

According to Lozowski et al. (1996) the lack of adequate explanation facilities becomes especially serious when, for instance, a neural network-based decision-making system is supposed to replace a human expert. Human experts can typically formulate rules that describe underlying causal relationships involved in the decision-making process. Neural network-based decision aids do not inherently possess such a key feature. Moreover, neither synaptic weights, activation levels, nor hidden layer responses can typically be meaningfully interpreted (Zurada 1992), since internal mappings emerge as superimposed intertwined relationships (Lozowski et al. 1996).

Compared to the amount of literature on theory and applications of neural networks, information on the analysis of neural networks is scarce (see Figure 3.1) and mostly limited to either *sensitivity analysis* or *rule extraction*. Sensitivity analysis (Rios Insua 1990) is commonly used in nonparametric statistical analysis, and is occasionally applied to neural networks. More often it appears in other mathematical modeling and decision-making systems. Rule extraction (Andrews et al. 1995) from neural networks originates from the field of symbol processing methods, which is rule-based rather than case-based. Neural network behavior described in sets of rules can provide insight into how the network comes to an answer. In Sections 3.1 and 3.2 these methods are treated in more detail. Literature also gives some alternative approaches. These are reviewed in Section 3.3.

In general, neural networks with unsupervised training, such as self-organizing feature maps, merely reorganize the input space by a winner-takes-all strategy. The output space, or feature vector space, is a reorganization rather than a transformation of the input space. However, this is not to be confused with the way in which the reorganization takes place. For example, if a self-organizing feature map reorganizes an $n$-dimensional input space ($n > 2$) onto a 2-dimensional lattice, this does not mean that the $n$-dimensional feature vectors (i.e., feature vectors with $n$ elements) are also reduced to 2-dimensional feature vectors (i.e., with 2 elements each). It means that the feature vectors are mapped onto a 2-dimensional map, whereby the mapping is based on a *selection* of properties of the feature vectors.

As an analogy, suppose a child is putting beads on a string. Initially, the 3-dimensional beads are stored in a 3-dimensional box. The child is now “mapping” the beads onto a 1-dimensional “map”, being formed by the string (in this example, we disregard the fact that the string is also a 3-dimensional object, as we are merely interested by the 1-dimensional lattice that is formed by the string). The beads remain 3-dimensional objects, even after they are put on the 1-dimensional string.

This mapping of feature vectors means that a comprehensible inspection of such neural networks after training becomes fairly simple: an investigation into the reorganized input space reveals how the network has restructured the input space. Section 2.6.1 already provided an example of this.

Analysis of neural networks trained under supervision, for example in the case of feedforward networks, is far more complicated, as they operate by non-linear vector manipulation. Whereas unsupervised neural networks *select*, supervised neural networks such as the feedforward networks *compromise*. In self-organizing feature maps the mapping of feature vectors is based on a selection of features (within the feature vectors) in order to reduce the dimensionality of the lattice in which they are initially situated. In feedforward networks new (internal) features are constructed by compromising the input features. These sets of internal features are not easily related to the input features. The purpose of an analysis is thus to find the proper relation between the internal features and the input features.

The following sections list several existing techniques for the analysis of neural networks. Advantages and disadvantages of the different techniques are related to their degree of applicability in a wide range of neural network application fields.

## 3.1 Sensitivity analysis

Sensitivity analysis (e.g., Rios Insua 1990) is a technique from the field of nonparametric statistical analysis. It is occasionally applied to neural networks, but more often in other mathematical modeling and decision-making systems. The general idea of sensitivity analysis is to investigate the effect that perturbations in the inputs have on the outputs, thus determining the sensitivity of the outputs to the inputs (Klimasauskas 1991). This is usually done for every input and every output, resulting in a matrix of sensitivities. These can then be used to determine whether any insignificant inputs can be ignored. In the neural network case, if a sensitivity analysis is performed for the hidden neurons, it could be used for pruning if some hidden neurons appear to have no influence on the network outputs (Engelbrecht and Cloete 1996). Fu and Chen (1993) use sensitivity analysis to investigate the generalization capability and error-correcting property of a neural network. Other examples of the use of sensitivity analysis for neural networks can be found in work by Choi and Choi (1992) and by Hashem (1992).

A weakness of sensitivity analysis applied to neural networks is formed by the fact that many applications of neural networks operate in high-dimensional input spaces, which would result in large sensitivity matrices that are hard to interpret. Another drawback is caused by the often strong nonlinearity of neural networks. This can cause outputs to have many different sensitivities over the full range of particular inputs, which in turn complicates the determination of the minimal value representation width or crisping (Blattner et al. 1993). Different analysis tools are needed in order to be able to analyze neural networks with high-dimensional input or output spaces or with strong nonlinear behavior. In fact, most neural networks have one or both of these properties, as they form some of the key advantages of neural networks. The cases in which sensitivity analysis could be applied to neural networks are mainly those cases where

- the networks are small, i.e., they have a limited number of internal parameters (degrees of freedom) that account for nonlinearities;
- sensitivity analysis is only applied to a (small) part of the neural network, such as one neuron or (a part of) a layer; or
- the dimensionality of the network has been significantly reduced, meaning that input and output spaces are low-dimensional, which is equivalent to the network having a low number of inputs and outputs.

In practice, these cases rarely occur. For that reason, sensitivity analysis is generally regarded not suitable as a tool to extract the knowledge that is stored in neural networks.

## 3.2 Rule extraction

Rule extraction from neural networks (e.g., Andrews et al. 1995) originates from the field of symbol processing methods, which is rule-based rather than case-based. Neural network behavior described in sets of rules can provide an insight into how the neural network comes to an answer. Craven and Shavlik (1994) distinguish two approaches to testing rules for multilayer neural networks: the *decompositional* approach and the *pedagogical* approach (validity-interval analysis (Thrun 1993)). The decompositional approach is to extract rules for each hidden and output unit separately, thus providing a certain transparency, whereas the pedagogical approach enables the extraction of rules that directly map inputs to outputs for a network as a whole, thus basically not opening the "black box." Pedagogical techniques are typically used in conjunction with a symbolic learning algorithm and the basic motif is to use the trained neural network to generate examples for the learning algorithm.

Andrews et al. (1995) classify rule extraction techniques into the following categories:

- Boolean rule extraction using decompositional approaches;
- Boolean rule extraction using pedagogical approaches;
- Extraction of fuzzy rules.

Due to its nature, Boolean rule extraction is mainly used in problems with discrete-valued features. Fuzzy rule extraction can be applied to both problems with real-valued features and those with discrete-valued features. Crisping is required to quantize the value space, but such may shudder the carefully constructed balance, that comprises the internal knowledge storage (Blattner et al. 1993).

Lozowski et al. (1996) use an algorithm to obtain rules in the form of logical implications which roughly describe the neural network mapping of a neural network-based decision-making system. In the case of rule extraction, the aim should be to extract knowledge from the neural network, not to replicate the mapping, since it is more important that the rules be as compact as possible even if classification with such rules is less than optimal. If a replication of the neural network is required, a rule base would quickly become very large, even for small networks, in order to still be able to account for all possible input situations.

Another indication for the limitation of the applicability of rule extraction can be found in the benchmark problems that are available for testing of rule extraction algorithms. Benchmark domain problems collected by Neumann (1998) can be solved by relatively easily trained very small networks. She lists the following benchmark problems:

**The monk's problems** (Thrun 1991) concerns three binary classification tasks involving six attributes taking discrete and binary values.

**The iris plant database** (Blake and Merz 1998) is a simple classification problem containing 50 instances each of three classes of iris plants. Each instance is described by four attributes taking continuous values. Setiono and Liu (1995) report high accuracies while using very small feedforward neural networks with only two hidden neurons.

**The Wisconsin breast cancer database** (Blake and Merz 1998) contains examples from two classes, each described by nine continuous-valued attributes. For this classification task, Setiono and Liu (1995) again report accurate classification with a pruned network containing just two hidden neurons.

**The DNA promotor domain** (Towell 1991) covers the real-world problem of finding genes in a DNA sequence. Each attribute in the 57-element input sequences takes on one out of five possible discrete values. Craven and Shavlik (1994) report that perceptrons (i.e., single-neuron "networks") are able to learn this binary classification task.

Neumann claims that all of these benchmark problems can be described with considerable accuracy by a small set of rules ($< 10$). She states that although these benchmark problems permit the test and comparison of accuracy and fidelity of extracted rules, using these benchmark problems does not provide information about the quality of the rules extracted and the complexity of the method if applied to networks for large real-world problems usually containing several hundreds or thousands of neurons.

To conclude, there are problem domains where solutions cannot easily or comprehensively be described in sets of rules or decision trees, due to, e.g., high dimensionality of the input space or very large sets of independent features. In those domains, different tools for analysis are needed, either in conjunction with rule extraction or separately.

## 3.3 Other methods

Literature offers a few examples that cannot directly be placed under sensitivity analysis or rule extraction. Feng et al. (1991) uses multilayer perceptrons for parameter estimation and also takes a first step to analyze the hidden units. Worth and Spencer (1989) describe a neural network for tactile sensing. After learning, the weight vectors of all hidden units have the same structure, but the authors did not find any correlation between the relative sizes of the weight vectors and the outputs of the hidden units. Van der Steen et al. (2001) indicate that the correlation between weight changes provides more information.

Mitchell (1992) presents a method for interpreting the role of the hidden neurons geometrically, when using neural networks for function approximation problems, in terms of regions of the input space. The relationship between this interpretation and the weights of the network connections is highlighted. For the case in which the approximation is of most interest over a bounded region of the input space, the author shows that this interpretation may be used to check for redundancy among hidden units, and to remove any such units found.

Olden and Jackson (2002) propose and demonstrate a randomization approach for statistically assessing the importance of axon connection weights and the contribution of input variables in neural networks for gaining knowledge of the causal relationships driving ecological phenomena. Their approach provides the ability to eliminate null-connections between neurons whose weights do not significantly influence the network output (i.e., predicted response of a variable), thus facilitating the interpretation of individual and interacting contributions of the input variables in the network. Furthermore, their randomization approach can identify variables that significantly contribute to network predictions, thereby providing a variable selection method for neural networks. In this sense, it resembles a sensitivity analysis, although it appears to be more widely applicable than sensitivity analysis.

Other analysis methods found in literature are basically nothing more than direct visualizations of the weights in the network. As such, they only have a certain degree of usefulness in small networks. One such method is the *Hinton diagram* introduced by Hinton et al. (1984) (or see Rumelhart et al. 1986b). In a Hinton diagram, synaptic weights of the neurons are represented by squares, where the relative sizes of the squares represent the magnitudes of the weights and the squares' colors (black or white) the polarities of the weights. Every neuron has a column of squares in the diagram, so the diagram is basically a table showing the values of the synaptic weights in a graphical manner, as can be seen from Figure 3.3. The Hinton diagram shown here represents the 3-neuron network shown in Figure 3.2.

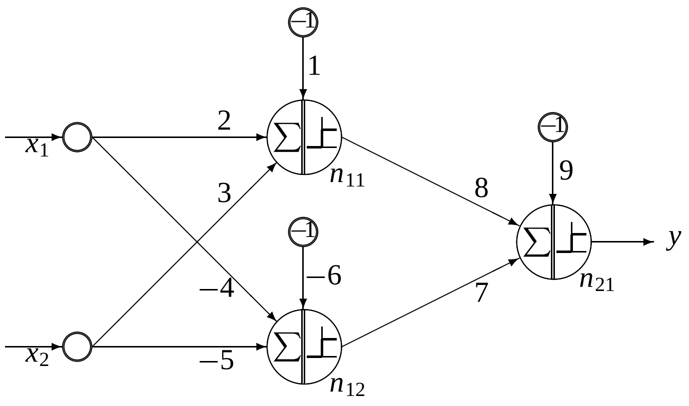


Figure 3.2: A feedforward neural network with two inputs, two hidden neurons, and one output neuron. Synaptic weights are shown along the connections between the units. Using threshold activation functions, this network performs the logical XOR operation on the Boolean inputs: $y = 0$ if $x_1 = x_2$ and $y = 1$ if $x_1 \neq x_2$.

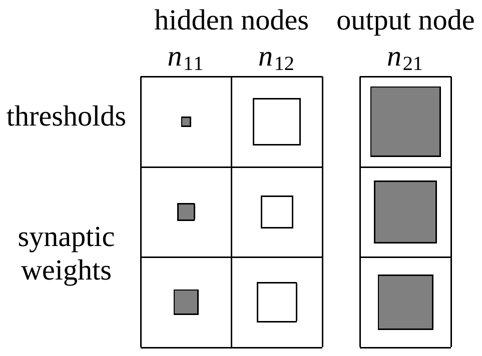


Figure 3.3: A Hinton diagram graphically showing the synaptic weight values of the neural network of Figure 3.2.

Another method suitable for smaller neural networks is the *bond diagram* introduced by Wejchert and Tesauro (1991). As can be clearly seen from Figure 3.4, the bond diagram preserves the structure of the network, and replaces the simple connections by “bonds.” The length of a bond represents the weight of the connection, and the color its polarity. Even though the bond diagram, like the Hinton diagram, basically just replaces numbers with their graphical representations, it does offer more insight into the network structure than the Hinton diagram, as the latter does not clearly show how the neurons are interconnected. Nevertheless, both diagrams have the disadvantage that they are only suitable for small networks, and they do not make clear what knowledge is represented by the synaptic weights.

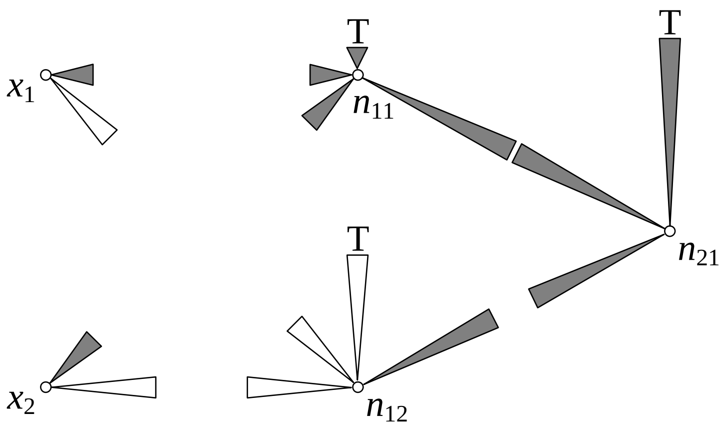


Figure 3.4: A bond diagram graphically showing the synaptic weight values of the neural network of Figure 3.2.

## 3.4 Conclusion

So far, mainly sensitivity analysis and rule extraction methods have been used to analyze neural networks. However, the previous sections make clear that these can only be applied in a limited subset of the problem domains where neural network solutions are encountered.

We therefor felt the need to investigate in which application domains trained neural networks cannot satisfactorily be analyzed with the knowledge extraction techniques discussed in this chapter. This is done for a selection of application areas in Chapter 4. Furthermore, we need to develop new methods to analyze trained neural networks in those domains.

This thesis therefore proposes a novel method for the analysis of neural networks. For a given problem domain, this new procedure firstly involves identification of basic functions with which users in that domain are already familiar. Then, the remainder of our procedure describes trained neural networks, or parts thereof, in terms of those basic functions. This provides a comprehensible description of the neural network's function. In other words, the knowledge that is implicitly present in the network is made explicit. Furthermore, depending on the chosen basic functions, it may also provide an insight into the network's inner "reasoning." The underlying theory is treated in Chapter 4, which also discusses the domain-dependency, and examples of its implementation can be found in Chapters 5 and 6.

Chapter 4

# Basic functions for neural network analysis

This chapter investigates for a selection of application fields in which of those areas trained neural networks cannot satisfactorily be analyzed with sensitivity analysis or existing rule extraction techniques. For those application domains for which no sufficiently satisfactory analysis methods are found, we will then explore the suitability of our analysis method based on domain-dependent basic functions, and give suggestions for basic functions that yield promising analysis results.

The novel analysis method proposed in this thesis involves two main steps. The first step is, for a given problem domain, to identify basic functions with which users in that domain are already familiar. The second step consists of describing trained neural networks, or parts thereof, in terms of those basic functions. This will provide a comprehensible description of the neural network's function and, depending on the chosen basic functions, it may also provide an insight into the network's inner "reasoning."

This concept is not so surprising, as many applications are based on a small number of arithmetic primitives. For instance, TerBrugge et al. (1995) describe the development of a stitch-weld tester. After a lengthy evaluation of various input features for the neural classifier, the acceptable solution is still reflecting the physical principle of operation: a dampened oscillation caused by reflections over different path lengths. In other words: it is based on a sine-wave and an exponential decay function.

Whether implicitly or explicitly, many application designs are based on the 3-tier model. The foundation layer in the 3-tier model contains the most basic functions for an application. These basic or primitive functions are usually a subset of a domain-wide set of primitive functions within the application domain. If a suitable subset of these domain-specific basic functions is chosen as the base for an analysis of a neural network application in that domain, the analysis is reduced to the task of finding a suitable mapping of the network parameters onto the basic functions.

Domain-specific analysis of neural networks through basic functions not only provides insight into the in- and external behavior of neural networks and shows their possible limitations in particular applications, but it also lowers the acceptability threshold for future users unfamiliar with neural networks. Further, neural network analysis methods that utilize domain-specific basic functions can be used to optimize neural network systems. An analysis in terms of basic functions may even make clear how to (re)design a superior system using those basic functions. In that case, the neural network merely serves as a system design advisor. If a user does not want to trust a neural network for any reason whatsoever, he may still trust a non-neural system that would have been nearly impossible to construct without using a neural network as an advisor.

The idea of describing a trained neural network in terms of domain-specific basic functions was introduced and presented in a number of earlier publications (Van der Zwaag and Slump 2002, Van der Zwaag et al. 2002a, b, c, d, 2003a, b, d). The main part of this chapter is based on those publications.

For many problems in certain domains, such as linguistics and decision theory, the common, domain-dependent basic functions can be chosen to be *if–then* rules or decision trees, in which case the analysis reduces to traditional rule extraction. Table 4.1 lists a few more problem domains where neural networks have been successfully applied. For each of these domains, potential basic functions are presented. In Chapter 5 we show for one such domain, i.e., digital image processing, how our analytical method can overcome the impracticality of more common knowledge extraction methods. In the case of edge detection – a digital image processing technique – the user will be familiar with image filters, particularly 2-dimensional differential operators. Hence, a description in terms of these digital image operators will enhance the understanding of the neural net's functionality. Therefore, Chapter 5 will illustrate the analysis of feed-forward–error-back-propagation neural networks trained for edge detection. For one other application domain, namely the general domain of classification problems, knowledge extraction in terms of class prototypes is illustrated with a case study in Chapter 6.

Each of the sections that follow, will treat one specific application field. Per domain, we investigate the suitability of the analysis of feedforward neural networks in terms of domain-dependent basic functions.

Table 4.1: Some application domains with potential domain-specific basic functions.

| application domain | potential basic functions | see section |
|---|---|---|
| decision theory | (fuzzy) if-then rules | 4.1 |
| signal processing (1-D) | operational filters | 4.2 |
| digital image processing (2-D) | differential operators | 4.3 |
| general classification problems | class prototypes | 4.4 |
| control theory | PID controllers | 4.5 |
| chemical process identification | molecular formulae | 4.6 |

## 4.1 Decision theory

In decision theory, neural networks have often been applied. There is also a growing interest in the analysis of neural network decision makers. This application domain is well suitable for the application of rule extraction methods such as described in literature (e.g., Andrews et al. 1995, Craven and Shavlik 1994). These methods usually result in rule bases or decision trees and hence result in descriptions of neural networks that should be easy to understand for human decision making experts.

In other words, it seems that this domain is already well-catered for, in the sense that classic rule extraction methods are in general very suitable for the analysis of neural network decision makers. However, we will show here that classical rule extraction can be regarded as a special case of analysis in terms of domain-dependent basic functions.

It seems obvious that the most suitable basic function for this domain is a generic *IF-THEN* rule. Using normal rule extraction techniques, the result of the translation of the neural network into a set of domain-dependent basic functions, i.e., a set of *IF-THEN* rules, is a rule base. This rule base can be in a tabular shape, or in the shape of a decision tree.

From the above follows that another, similar, candidate for basic functions in this domain is formed by segments of decision trees. These would basically just be a different representation of *IF-THEN* rules, with a similar outcome in the shape of a decision tree. For example, Schmitz et al. (1999) grow binary decision trees from trained neural networks with the help of an attribute selection criterion that is based on significance analysis.

In conclusion, we can state that on the one hand there is no direct benefit of our proposed method for the application field of decision theory. On the other hand, the analysis techniques that have been successfully applied to neural networks decision making systems, are in fact already realizations of analysis in terms of domain-dependent basic functions.

### 4.1.1 An application

Rule extraction techniques for neural network decision making systems can be readily found in literature. For instance, Lozowski et al. (1996) have developed and tested a neural network decision system for a medication dosage problem to demonstrate their approach for the extraction of rules from a trained neural network. Their rules have a form of those used in fuzzy reasoning. Furthermore, they state that the neural network is a necessary element if the training data is noisy, since the neural network provides filtration of the training data and thus the number of conclusive rules becomes reasonable. Even one noisy data point would cause an enormous increase in the number of created rules if the rules were obtained based on the training data instead of the network outputs.

In their paper, Lozowski et al. introduce an algorithm to obtain rules in the form of logical implications which roughly describe the neural network mapping. The number of extracted rules can be selected by using an uncertainty margin parameter as well as by changing the precision of the soft quantization of the inputs. The number of rules is related to the accuracy of a fuzzy classifier using the created rules. According to the authors, the aim is to extract knowledge from the neural network, not to replicate the mapping, since it is more important that the rules be as compact as possible even if classification with such rules is less than optimal.

In order to test their rule extraction algorithm, Lozowski et al. trained a neural network with three inputs and one output. The input features were quantized into three fuzzy classes for the sake of fuzzy rule extraction. This led to rule sets consisting of a maximum of 27 rules. Totally undecidable rules were subsequently pruned from the rule sets. As an example, the authors presented an extracted rule base with 12 rules. In Table 4.2 an example of a full rule base is given, which can be reduced to the more concise rule base shown in Table 4.3 by pruning indecisive rules and combing rules where possible (N.B., these rule sets are not the result of an actual rule extraction, but merely serve an illustrational purpose). However, the rule base in Table 4.3 is not complete, unless the default decision for unlisted inputs would be defined as “indecisive”.

One can imagine that a quantization into more fuzzy classes or a system with more inputs would inevitably lead to a (much) larger extracted rule base, if a similar rule

Table 4.2: Example of a rule base containing 27 rules.

IF $p_1$ is $(p_1)$ AND $p_2$ is $(p_2)$
AND $p_3$ is $(p_1)$ THEN $r$ is $(r)$

| $p_1$ | $p_2$ | $p_3$ | r |
|---|---|---|---|
| *low* | *low* | *low* | *medium* |
| *low* | *low* | *medium* | *medium* |
| *low* | *low* | *high* | *medium* |
| *low* | *medium* | *low* | *high* |
| *low* | *medium* | *medium* | *medium* |
| *low* | *medium* | *high* | *medium* |
| *low* | *high* | *low* | *high* |
| *low* | *high* | *medium* | *high* |
| *low* | *high* | *high* | *medium* |
| *medium* | *low* | *low* | *medium* |
| *medium* | *low* | *medium* | *medium* |
| *medium* | *low* | *high* | *low* |
| *medium* | *medium* | *low* | *medium* |
| *medium* | *medium* | *medium* | *medium* |
| *medium* | *medium* | *high* | *medium* |
| *medium* | *high* | *low* | *indecisive* |
| *medium* | *high* | *medium* | *high* |
| *medium* | *high* | *high* | *medium* |
| *high* | *low* | *low* | *medium* |
| *high* | *low* | *medium* | *low* |
| *high* | *low* | *high* | *low* |
| *high* | *medium* | *low* | *medium* |
| *high* | *medium* | *medium* | *medium* |
| *high* | *medium* | *high* | *low* |
| *high* | *high* | *low* | *medium* |
| *high* | *high* | *medium* | *medium* |
| *high* | *high* | *high* | *indecisive* |

extraction algorithm would be applied to that system. In fact, many neural network applications in other application fields do indeed have more inputs or would indeed require more fuzzy classes in order to have an acceptable level of accuracy. In those cases, it is advisable to use different basic functions, tailored to the domain in question.

Table 4.3: Example of a reduced rule base containing 12 rules.

IF $p_1$ is $(p_1)$ AND $p_2$ is $(p_2)$
AND $p_3$ is $(p_1)$ THEN $r$ is $(r)$

| $p_1$ | $p_2$ | $p_3$ | r |
|---|---|---|---|
| *medium* | *low* | *high* | *low* |
| *high* | *low* | *¬low* | *low* |
| *high* | *medium* | *high* | *low* |
| *low* | *low* | - | *medium* |
| *¬low* | *low* | *low* | *medium* |
| *low* | *medium* | *¬low* | *medium* |
| *medium* | *low* | *medium* | *medium* |
| *medium* | *medium* | - | *medium* |
| *high* | *¬low* | *¬high* | *medium* |
| *¬high* | *high* | *high* | *medium* |
| *low* | *¬low* | *low* | *high* |
| *¬high* | *high* | *medium* | *high* |

## 4.2 Signal processing

There exists a considerable amount of literature on applications of artificial neural networks in the field of signal processing, see, for example, the proceedings of the *IEEE Workshops on Neural Networks for Signal Processing* (e.g., Principe et al. 1997). However, it is difficult to find literature on the extraction of knowledge from neural networks for 1-dimensional signal processing, including systems for time series prediction.

A method to extract knowledge from recurrent neural networks for financial forecasting was introduced by Giles et al. (2001). Symbolic knowledge was extracted from the trained recurrent neural networks in the form of deterministic finite-state automata. According to the authors, these automata explain the operation of the system and are often simple. Automata rules related to well-known domain-specific behavior, such as trend following and mean reversal, were extracted. Although the authors report positive rule extraction results for the financial forecasting problem that they considered, one should realize that in recurrent neural networks the temporal relationship of a time series is modeled in an explicit manner via the internal states of the network. This is typically not the case for feedforward neural networks. Nevertheless, the idea of using deterministic finite-state automata as domain-specific functions for the application area of signal processing in general and (financial) forecasting in particular, seems promising.

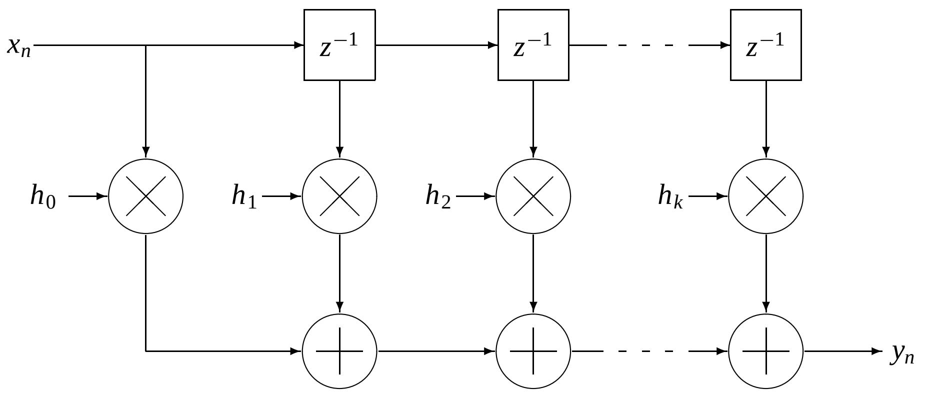


Figure 4.1: Block diagram of a digital FIR filter.

Several other options are available for choosing basic signal processing functions. As filters are key elements in most signal processing systems, it seems logical to use filters as domain-specific basic functions for the signal processing domain. For instance, finite impulse response (FIR) filters form a generic class of filters that seem ideal for this purpose. An schematic representation of a digital FIR filter is given in Figure 4.1. More formally,

$$y_n = \sum_{k=0}^{N-1} h_k x_{n-k}. \tag{4.1}$$

For a description of a neuron in terms of a FIR filter, the synaptic weights of the neuron need to be translated into the coefficients $h_k$ of the FIR filter. These coefficients can then be shown graphically (as a discrete impulse response function), to enable easier interpretation. If each neuron is translated into a FIR filter, it will be possible to describe the whole network as a filter bank.

Other candidates for basic functions in signal processing are gradient filters. Every neuron can be described as a set of gradient filters of different order. Whereas in the application field of (1-dimensional) signal processing gradient filters are one-dimensional, in the field of digital image processing (i.e., 2-dimensional signal processing) the gradient filters are of course 2-dimensional. The next section looks at the application domain of image processing.

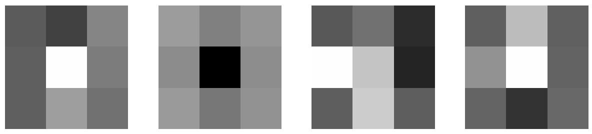


Figure 4.2: Graphical representation of the synaptic weights of the hidden neurons of a neural network edge detector.

## 4.3 Digital image processing

Artificial neural networks are frequently applied in the field of digital image processing. However, when feedforward–back-propagation neural networks are used, the network typically remains a closed black box. Occasionally, the box is opened by means of graphically representing the synaptic weights of the neurons in some way or another, but it remains quite dark inside the box. An example of such a graphical representation can be seen in Figure 4.2, which visualizes the synaptic weights of the hidden neurons of a neural network edge detector. Experts in the field of digital image processing may still be able to derive some properties from the graphs, but they will not be able to give a detailed description of the function performed by the neurons or of that performed by the network as a whole.

Here it seems clear that major improvements can be made if an alternative, more accurate analysis method would be available.

In Chapter 5 this application field is widely explored and an accurate and comprehensible analysis method is introduced based on the theory of neural network analysis in terms of domain-dependent basic functions.

## 4.4 Classification in general

Many neural network solutions have been developed in classification problems. As in most application domains, it is difficult to gain insight into the internal functionality of feedforward neural networks trained to perform some classification task. Direct inspection of the parameters of the network, the weights of the internal connections, does not reveal the knowledge that is present in the network, at least not in a human-understandable way. This is much less the case for, e.g., nets that are trained unsupervised, such as Kohonen's self-organizing feature maps (Kohonen 1982), as shown earlier in Section 2.6.1. These offer a direct view into the stored knowledge, as their internal knowledge is stored in the same format as the input data that was used for training or is used for evaluation.

Even for feedforward networks, many successful attempts have been made to accurately describe the classifying function that has been learned by neural networks. However, these are almost always based on testing the neural network classifier with test samples until a satisfactory class overview has been found. It is therefor an improvement if the class overview can be directly derived from the neural network rather than indirectly through the use of test samples.

In general, classification problems can also be considered as decision making systems. For every class present in the output space and for each input pattern offered to the system inputs, the system needs to make a decision regarding whether the pattern belongs to that class or not. Thus, the ideas expressed in Section 4.1 regarding decision making systems are also valid for (small) classification systems. In this context we can note an example from literature where a neural network that was trained for a classification task is translated in a set of rules. Palade et al. (2001) trained a feedforward network with three hidden neurons for classification of the well-known benchmark iris data set (Blake and Merz 1998). Crisp rule extraction led to a rule base containing 150 rules, which covered 94% of the network functionality. A reduction to 25 rules covered just 42% of the network functionality. The authors state that, in order to cover almost the entire network functionality, the obtained number of extracted rules was very much increased, showing one of the main drawbacks of approximation techniques by traditional crisp rules. In a second experiment, Palade et al. (2001) extracted fuzzy rules, with which they were able to much more concisely represent the neural network. However, the fuzzy rules themselves were less comprehensible than the crisp rules from the first experiment.

As mentioned before in Section 4.1, crisp or fuzzy rule extraction is mainly practical for smaller neural networks. So, for classification systems, we propose a method to represent the knowledge that is stored in the trained neural network through the extraction of class prototypes. In Chapter 6 this idea is worked out as a case study.

## 4.5 Control theory

Artificial neural networks are frequently applied in control systems. However, literature on the analysis of trained neural network-based control systems is scarce. For small neural systems, fuzzy rule extraction can be applied. The rule bases that will be the result are probably similar to the rule bases used in fuzzy control systems.

For larger neural network-based control systems, we propose the use of proportional-integral-derivative (PID) controllers as domain-dependent basic functions for the control domain. No examples of an analysis of neural network-based control systems in terms of PID controllers has been found. However, a related subject is neural network control systems based on PID controllers. Scott et al. (1992) presented the MANNCON (Multivariable Artificial Neural Network Control) algorithm by which the mathematical equations governing a PID controller determine the topology and initial weights of a neural network, which was further trained using back-propagation. They applied this method to the task of controlling the outflow and temperature of a water tank, producing statistically-significant gains in accuracy over both a standard neural network approach and a non-learning PID controller.

Scott et al. (1992) finish their paper with the observation that neural networks are generally considered to be "black boxes," in that their inner workings are completely uninterpretable. Since the neural networks in their approach were initialized with information, the authors think it may be possible to interpret the weights of the network and extract useful information from the trained network. However, they make no attempt at actually trying to extract knowledge from the trained network.

Nevertheless, it seems promising to explore the possibilities of translating neural-network-based controllers into PID controllers (Stephanopoulos 1984), which form one of the simplest of the traditional feedback controller schemes. The basic idea of a PID controller is that the control action should be proportional to (a) the error, (b) the integral of the error over time, and (c) the temporal derivative of the error (hence the term proportional-integral-derivative controller). The contribution of the various components is determined by several tuning parameters. The task of an algorithm for the analysis of trained neural network-based control systems in terms of PID controllers would then be to identify suitable tuning parameters based on the synaptic weights of the neural network.

Tan et al. (1999) proposed and implemented a generalized nonlinear PID controller that is based on neural networks. They used a gradient descent algorithm to train the neural network-based control system. The tuning degree of the algorithm was increased in order to make it possible to find a group of tuning parameters to obtain a satisfactory training performance. The derived convergent condition for their algorithm provides an effective way for choosing proper PID parameters leading to stable and fast convergence.

## 4.6 Process/system identification

Process identification is a major part of control theory, where the (partly) unknown process must be monitored and modeled before it can be controlled. This has been pursued through numerous mathematical schemes (Trentelman and Willems 1993) but also through fuzzy logic (Lam et al. 2002) and neural networks (Narendra and Parthasarathy 1990). The observations on the process (temperature, sound, vibration, flow, etc.) are assumed to become available as signals, though lately also (spectral) images have come into use.

From centuries of scientific research, some basic knowledge on the process behavior is available as natural laws. Years of experience in operating the process also produces a degree of understanding: the operator knowledge. Where this has not been made operational by scientific research, fuzzy modeling provides a means to capture such knowledge for analysis and simulation.

However, processes tend to change over time. Hence a lengthy analysis based on the single set of historic measurements will fail to properly model the current situation. The model must be formally proven to be robust or periodically refreshed by observing the measurement data. It has been shown in various publications, that neural networks can be used to capture reality. However, neural networks may be hard to learn and combinatorial difficult to understand. Hence neural networks are often only used to provide operator assistance on a strategic level.

As in the previous section, this section will also approach the localization of knowledge from a constructive point-of view. The main part of this section has been published in earlier work (Van der Zwaag et al. 2002c).

This section sets out to explore the possibilities to provide in-line assistance to the process modeler. It is based on advances in two directions. Neural networks are used to capture the process knowledge in a fusion of existing knowledge and measured data. Such networks are modular to guarantee learnability and to allow for knowledge re-use. Using a large number of small nets in stead of a single one suppresses the combinatorial explosion of the search space from which the rule extraction on monolithic neural networks suffers. This can be further supported by the introduction of basic functions: domain-specific pieces of common truth that help to construct the model by reading from a heterogeneous network with frozen parts.

### 4.6.1 Modular neural networks

Compositions of neural networks have been studied for some time now to increase the capacity and accuracy of neural networks (Auda and Kamel 1999). Though claimed to provide a better performance, the fundamental difficulty in network training (Barakova 1999) remains a major concern and several studies have been made for solutions to this problem.

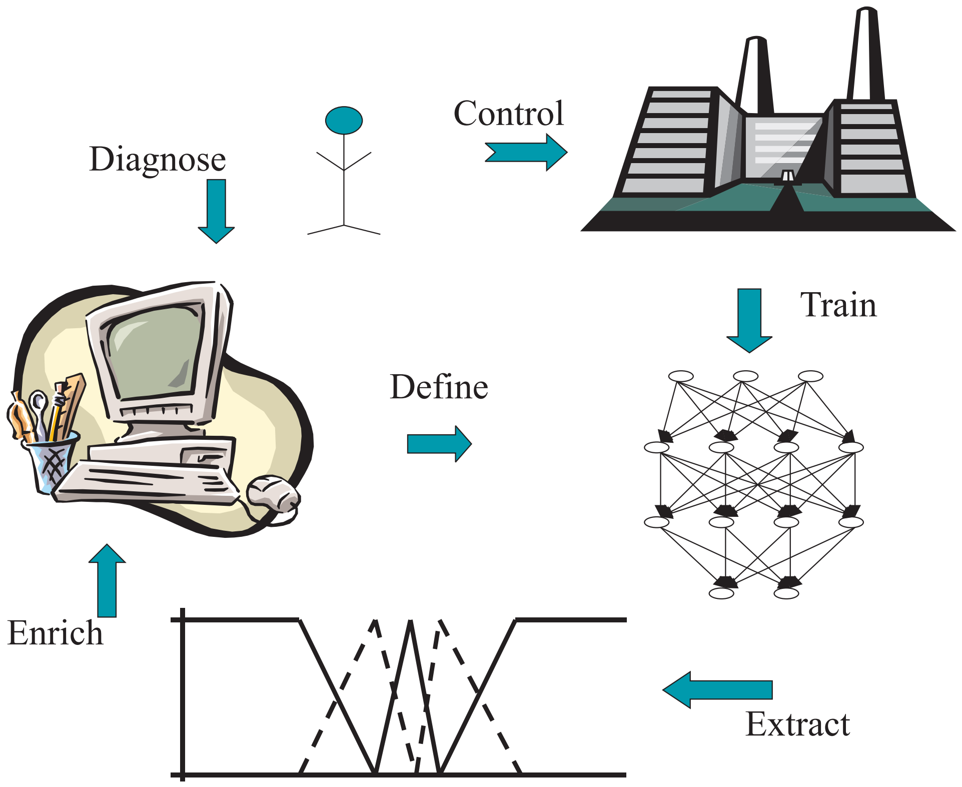


Figure 4.3: The knowledge–in–the–loop architecture is based on the iterative improvement of the process model through the extraction of the knowledge within the (post-)training neural network.

Modular networks typically consist of a network of interconnected neural networks (also called modules or rule blocks) that each solve a sub-problem (Spaanenburg et al. 1997a). For this approach, extensions to the standard back-propagation learning algorithm are necessary; for instance, techniques to schedule the learning of the modules to prevent unlearning, as proposed by Spaanenburg (2000).

Hierarchical networks go one step further in the compositional sense (Keegstra et al. 1996). Each node in a neural network may again be a neural network, or a specialized function. So a node's evaluation function is implemented by a neural network, while preserving the weights on its inputs from the upper layer. It is sometimes difficult to distinguish these two notions, as a modular network may have a hierarchical construction while a hierarchical construction may locally use a modular function.

The major benefit of modular networks follows from the inherent *separation of* *concerns*. The disreputed failure to learn in monolithic neural nets is caused by the presence of conflicting features, where the feed-forward arrangement tends to compromise instead of to select. During initial learning this leads to plateaus or local extrema in the error space; later on it may cause unlearning or catastrophic forgetting (Barakova 1999). Separating such conflicts and solving them in separate modules before assembling the overall network solves such problems. This is similar to the effect of feature preprocessing (TerBrugge et al. 1995), which reputedly involves more than 80% of the project effort.

Modular networks can be based on a natural partition of the problem space reflecting the domain knowledge. Where such knowledge is not fully available, but a linguistic description is feasible, fuzzy modeling can be applied. Using a multi-block fuzzy representation, the fuzzy model will have a structure that mimics the domain structure and which can block-by-block be transformed into a modular neural net (Spaanenburg et al. 1997a).

When the process deviates too much from the model, special gates and guards may be required to check whether a module remains within the assigned validation area. In a *gating network*, each module receives only the data within the assigned validation area, whereas in a *guarded network* it receives all data but it is monitored whether the module behavior does not deviate too much from the expectations (Venema 1999).

Spaanenburg et al. (1997b) monitor the errors on the signals between the modules. Each module is trained in isolation and connected when the inter-module error becomes small. Vice versa, when the network is post-trained and the error appears to grow, the connection is broken again. This method requires that the error size can be qualified as being “too large.” This is not always easy to do.

It has already been observed (Barakova 1999) that a conflict-free neural network has a stable and reproducible learning behavior. Similar observations while attempting to map neural networks on small micro-controllers have led to the understanding that irreproducible behavior is caused by a re-orientation of the neural network in its selection of hidden features (Spaanenburg et al. 2001). When a neural network has to change the selection of hidden features in order to optimize the output function, this takes considerably more learning time than when it merely has to adapt the output weights.

Such has two consequences. The first is that learning time can be used as a measure to activate modules during training (Venema and Spaanenburg 2001). In a time-ordered schedule the modules start to learn while being part of the overall network; the timing of activation is decided on the momentary learning times per module (Figure 4.4). The second consequence is that abnormal behavior can be detected by

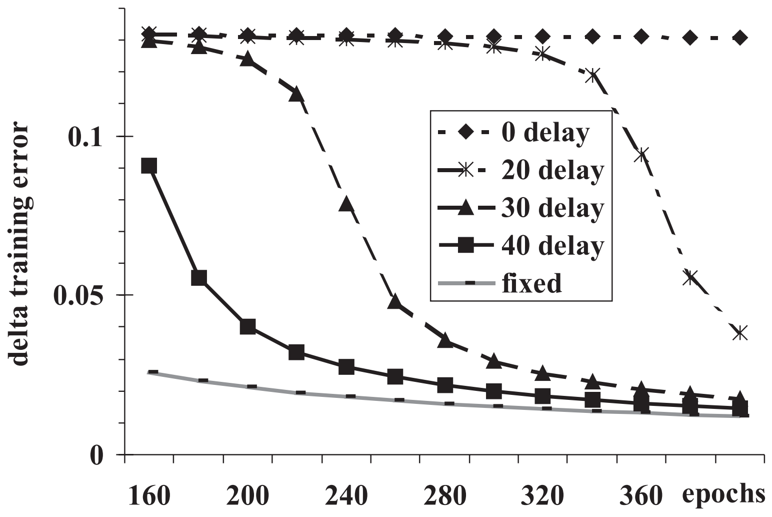


Figure 4.4: Impact of a small delay in the start of module activation, according to Venema and Spaanenburg (2001).

monitoring the learning time (Spaanenburg 2001). This is shown by vanVeelen et al. (2000) for novelties in process identification on the network level, but can also be applied on the module level. This measure provides therefore an easy means to guarantee model integrity during the (post)learning of autonomous functions such as agents. But such networks are still hard to prove correct by the lack of documentation on the stored knowledge.

### 4.6.2 Domain development

Many production processes in the heavy industry are based on a judicious mixture of physical and chemical receipts. It is the shady world between molecular physics, metallurgy and mechanical engineering. Such processes are not always easily understood, let alone modeled. It is not that the fundamental knowledge is missing but rather that the dimensions of reality are not easily accounted for. More often than not the production line must be carefully adapted and tuned by an interdisciplinary team.

The 3-tier modeling technique has become very popular in modern computing science as it allows to match the needs of the application to the promise of the technological fundament. Of course, nature is too complicated to be summarized in merely three layers. Comparatively simple things as the ISO/OSI digital network model take already a 2-level hierarchy of 3-tier composites. We conjecture that the

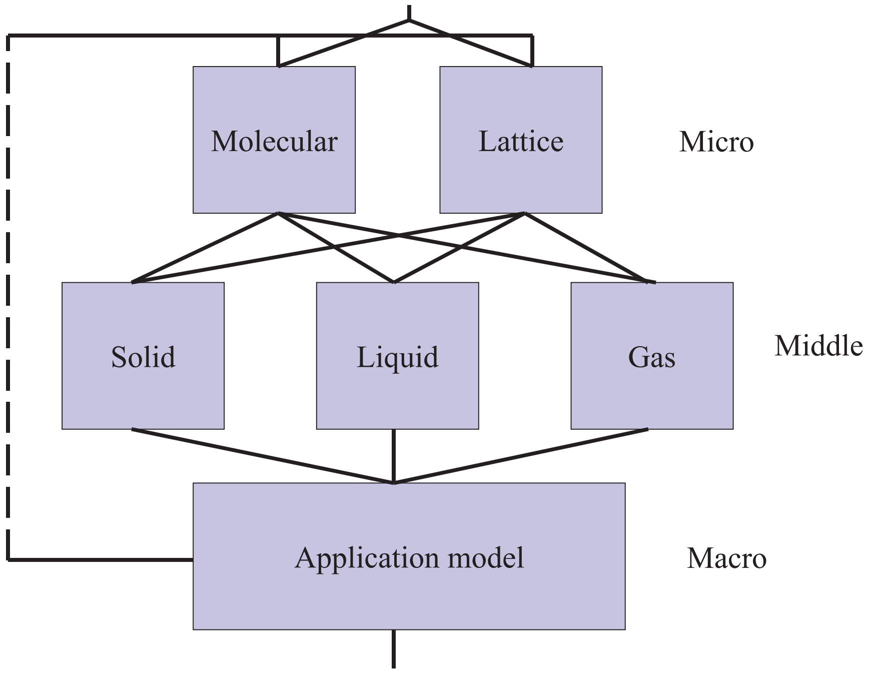


Figure 4.5: The process domain model.

modular, physically plausible 3-tier can be iteratively detailed by a hierarchy of self-similar 3-tiers to handle a higher problem complexity.

A suitable domain model for production processes can be defined along the lines of the 3-tier concept. The basic rules emanate from micro-corpuscular considerations, where the attractive and distractive forces between the atomic elements are handled. The macro-corpuscular view is an abstraction thereof but this relation is of a stochastic nature. Where the present is not a mirror of the past, the predictive power of stochastics is sometimes overrated. In turn, the macro-corpuscular view can be abstracted to application-oriented phenomena. Such a domain model is shown in Figure 4.5.

The proper choice of basic functions allows to introduce existing knowledge rather than to start from blind learning. The micro layer uses existing molecular formulae as basic functions in a neural setting, because the precise formulation for the various phases lacks robustness. The application model is a regular neural network that reflects the itinerary through the phases and uses a simple sigmoid. The middle layer glues these two together and is based on conventional physical principles to relate pressure, time and volume. It uses the 2nd-order differential transfer function as proposed by Meijer (1996).

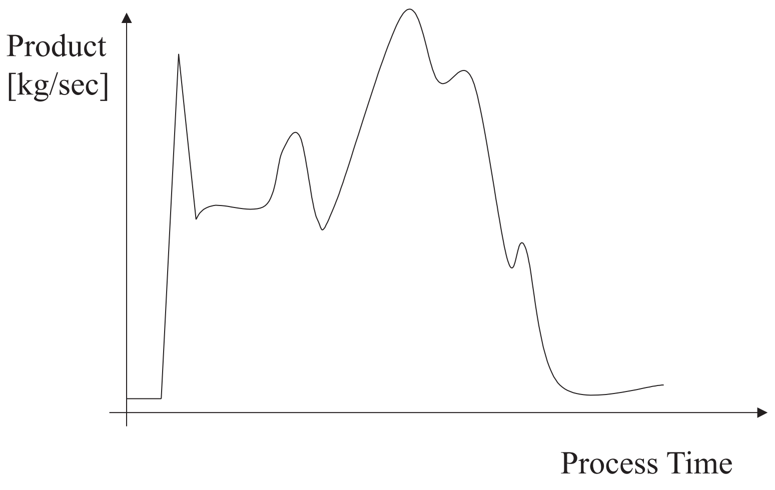


Figure 4.6: Typical indirect exhaust measurement.

Striking in the domain model is the role of the phases. The various ways in which matter can be present is usually taken as the first module layer. This is similar to the situation with weather prediction where the seasons are used to derive the prediction (Venema 1999). Such an architecture makes it hard to distinguish between a cool day in summer and a warm day in winter time; simultaneously transitional regions as spring and autumn become hard to handle. It has been shown that a first distinction between warm and cold days before defining seasons gives a better model. Along the same reasoning it only complicates things to start directly from the different phases to handle the various itineraries of the overall process and it is required to provide a proper base in molecular physics through the phase diagram.

The above process domain model has been suggested for the case of a metal scrap furnace, where the exhaust fumes are measured in controlling the process to optimize the industrial product and minimizing the environmental damage. Lack of facilities to measure directly on the process makes it necessary to bring all available information into one model and to enhance this model from experience (Figure 4.3).

A typical measurement is shown in Figure 4.6. The relation with the itinerary through the phase diagram is not immediately comprehensible, but the principle P-V-T behavior of metal scrap is known. This makes that the basic requirements for training the network are fulfilled and similarly shaped curves can be routinely handled. However, as the aggregation changes at seemingly arbitrary moments, the feedback in Figure 4.5 between the application and the molecular physics layer is mandatory to characterize the future exhaust level based on the itinerary drift.

The methodology will be most attractive for the characterization of production processes, where the nature and location of the process make a direct measurement of process variables difficult, if not impossible. In such cases, sensors like the camera or the audiometer can be used to measure non-electrical process parameters and carry them into the input domain where a neural net can operate.

# Chapter 5

# A case study in image processing

This chapter first recalls parts of the theory of digital image processing, more in particular from the subfield of pixel classification. After that it presents an example of a feedforward neural network that has been trained to detect edges in digital images. Finally, an accurate and comprehensible analysis method is presented based on the theory of neural network analysis in terms of domain-dependent basic functions.

This chapter is based on a number of earlier publications (e.g., Van der Zwaag and Slump 2002, Van der Zwaag et al. 2003b) and is a modification of a book chapter (Van der Zwaag et al. 2003a) that was published earlier this year (Jain et al. 2003).

## 5.1 Digital image processing

In this section we will first treat several basic techniques commonly used in digital image processing, or more specifically, in pixel classification. Edge detection (Section 5.1.1), line detection (Section 5.1.2), and spot detection (Section 5.1.3) are often a first step in "higher-level" techniques such as image classification, image segmentation, image enhancement, and others. Section 5.1.4 deals with the more generic class of differential operators, or gradient filters, which can also be used to detect edges, lines, and spots.

### 5.1.1 Edge detection

Edge detection is frequently used in image segmentation. In that case an image is seen as a combination of segments in which image data are more or less homogeneous. Two main alternatives exist to determine these segments: (1) classification of all pixels that satisfy the criterion of homogeneousness; (2) detection of all pixels on the borders between different homogeneous areas. To the first category belong pixel classification (depending on the pixel value the pixel is part of a certain segment) and region growing methods. The second category is edge detection.

In fact, edge detection is also some sort of pixel classification: every pixel is either part of an edge or not. All edges together form the contours of the segments. After edge detection sometimes edge linking is used, in order to try to get the contours closed, as in practice not all pixels will be classified correctly, due to noise, etc. Many edge detection filters only detect edges in certain directions, therefore combinations of filters that detect edges in different directions are often used to obtain edge detectors that detect all edges.

In digital image processing, we can write an image as a set of pixels $f_{n,m}$. An edge detection filter which detects edges with an angle $\phi$ (i.e., relative to a horizontal edge) can be written as a (template) matrix with elements $w_{k,l}$, see Figure 5.1 in Section 5.1.5, where this is further elaborated. We can then determine whether a pixel $f_{n,m}$ is an edge pixel or not, by looking at the pixel's neighborhood, see Figure 5.2 (Section 5.1.5), where the neighborhood has the same size as the edge detector template, say $(2K+1)\times(2L+1)$. We then calculate the discrete convolution

$$g_{n,m} = \sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l}\, f_{n-k,m-l}, \tag{5.1}$$

where $f_{n,m}$ can be classified as an edge pixel if $g_{n,m}$ exceeds a certain threshold and is a local maximum in the direction perpendicular to $\phi$ in the image $g_{n,m}$.

Some examples of templates for edge detection are:

$$\underset{\text{Sobel, }0^\circ}{\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}}, \quad \underset{\text{Kirsch, }45^\circ}{\begin{bmatrix} -5 & -5 & 3 \\ -5 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}}, \quad \underset{\text{compass, }90^\circ}{\begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix}}.$$

The dependency on the edge direction $\phi$ is not very strong; edges with a direction $\phi \pm 45^\circ$ will also activate the edge detector, though less strongly.

### 5.1.2 Line detection

A similar filter is the line detector, which detects lines rather than edges. In fact, a line can be seen as two edges lying parallel and close to each other. Actually, most line detectors will also detect some edges, and most edge detectors will also detect some lines. In practice, there is not a well-defined boundary between lines and edges. For example, one could think of a line between two image segments (a line on an edge), or a narrow image segment with edges on either side (how narrow would such a segment need to be in order to be regarded as a line rather than as a region with two parallel edges?).

Some simple line detector templates are:

$$\underset{\text{horizontal, }0^\circ}{\begin{bmatrix}-1&-1&-1\\2&2&2\\-1&-1&-1\end{bmatrix}},\quad \underset{\text{vertical, }90^\circ}{\begin{bmatrix}-1&2&-1\\-1&2&-1\\-1&2&-1\end{bmatrix}},\quad \underset{\text{diagonal, }45^\circ}{\begin{bmatrix}-1&-1&2\\-1&2&-1\\2&-1&-1\end{bmatrix}},\quad \underset{\text{diagonal, }-45^\circ}{\begin{bmatrix}2&-1&-1\\-1&2&-1\\-1&-1&2\end{bmatrix}}.$$

Combining these four templates would result in a detector capable of detecting lines in practically all directions. Only strongly curved or crossing lines might not be detected equally well.

### 5.1.3 Spot detection

A third filter type is the spot detector. A spot is an image segment approximately the size of a pixel. This of course depends on the image resolution; an object that shows as a spot in a low-resolution image would appear larger in a higher resolution, and an object that shows as a spot in a high-resolution image might very well be invisible in a lower-resolution image of the same scene.

Example templates for spot detection are:

$$\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix},\quad \begin{bmatrix}-1&0&-1\\0&4&0\\-1&0&-1\end{bmatrix},\quad \begin{bmatrix}-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1\\-1&-1&24&-1&-1\\-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1\end{bmatrix},\quad \begin{bmatrix}-1&-1&-1&-1&-1\\-1&1&2&1&-1\\-1&2&4&2&-1\\-1&1&2&1&-1\\-1&-1&-1&-1&-1\end{bmatrix}.$$

The templates on the right are slightly less sensitive to noise than the ones on the left, because of the difference in the neighborhood sizes. Nevertheless, these templates are quite sensitive to noise, which is easy to understand, given that noise often appears as spots. Because spots have no direction, the above templates are omnidirectional. This will be confirmed by the analysis results in Section 5.2.2.

Spot detectors can also be used to detect lines, because lines have similar properties as spots. Theoretically seen, spots have no width and no length, and lines have no width. However, whereas in Section 5.1.2 we needed a combination of several directional templates to detect lines in all directions, if would be sufficient to use only one omnidirectional spot detection template to detect lines in all directions. The reason why directional line detectors are often preferred lies in the larger noise sensitivity of the spot detection filters.

### 5.1.4 Differential operators

A more generic type of image filters are the differential operators. Usage of these operators is based on the detection of changes in greylevel in a greylevel image. The gradient vector of a 2-dimensional continuous image $f(x, y)$ is defined (e.g., by Van der Heijden 1994) as

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} & \frac{\partial f(x,y)}{\partial x} \end{bmatrix}^T = \begin{bmatrix} f_x(x,y) \\ f_y(x,y) \end{bmatrix}. \tag{5.2}$$

For discrete images this can be seen as a template $[-1\ 1]$ for the gradient in the horizontal direction and as a template $\left[\begin{smallmatrix} 1 \\ -1 \end{smallmatrix}\right]$ for the gradient in the vertical direction. The gradients in the diagonal directions can be determined with Roberts' templates $\left[\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right]$ for gradients in $-45^\circ$ and $+45^\circ$ directions, respectively. Note that the direction of the edges detected by a first-order differential operator is perpendicular to the direction of the gradient. The same holds for the lines detected by a second-order differential operator.

### 5.1.5 Describing a template in terms of differential operators

We will now describe how an arbitrary image filter in matrix form can be seen as a composition of several differential operators, where the operators are of varying orders and operate in varying directions. In order to transform a template into a set of gradient filters, we first calculate the Taylor series expansion of the Fourier transformed template, and then we apply the inverse Fourier transform to get a description of the template which gives us knowledge about the differential components. The reason for using a Fourier transformation lies in the fact that a Fourier transformed filter description consists of a series of sinusoidals, which are easily differentiated to determine the Taylor components.

$$\begin{bmatrix} w_{-K,L} & \cdots\cdots\cdots & w_{K,L} \\ \vdots & \ddots & \vdots \\ \vdots & w_{0,0} & \vdots \\ \vdots & \ddots & \vdots \\ w_{-K,-L} & \cdots\cdots\cdots & w_{K,-L} \end{bmatrix}$$

Figure 5.1: A $(2K+1)\times(2L+1)$ template $w_{k,l}$.

$$\begin{bmatrix} f_{-N,M} & \cdots & \cdots & \cdots & f_{N,M} \\ \vdots & f_{n-K,m+L} & \cdots\cdots\cdots & f_{n+K,m+L} & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & f_{n,m} & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & f_{n-K,m-L} & \cdots\cdots\cdots & f_{n+K,m-L} & \vdots \\ f_{-N,-M} & \cdots & \cdots & \cdots & f_{N,-M} \end{bmatrix}$$

Figure 5.2: A $(2N+1)\times(2M+1)$ image with a $(2K+1)\times(2L+1)$ neighborhood around $f_{n,m}$.

### The template in the spatial domain

In digital image processing, we can write an image filter as a (template) matrix (Van der Heijden 1994) with elements $w_{k,l}$, see Figure 5.1.

### The input image in the spatial domain

We can write an image as a matrix of pixels $f_{n,m}$, see Figure 5.2.

### Filter operation in the spatial domain

As already stated in Section 5.1.1, in pixel classification a filter operation can be defined by the discrete convolution of the filter template $[w_{k,l}]$ with the local neighborhood around a pixel $f_{n,m}$ in the input image $[f_{n,m}]$:

$$g_{n,m} = \sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l}\, f_{n-k,m-l}, \tag{5.3}$$

where $g_{n,m}$ is the output pixel when the filter template $[w_{k,l}]$ is applied to the input image pixel $f_{n,m}$.

**From discrete to continuous in the spatial domain**

To facilitate Fourier transformation, two-dimensional Dirac functions ($\delta$), defined as

$$\delta(x,y) = 0 \quad \text{for} \quad x^2 + y^2 \neq 0;$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(x,y)dx\,dy = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(0,0)dx\,dy = 1, \tag{5.4}$$

are assigned to the pixels $f_{n,m}$ and to the template elements $w_{n,m}$. This gives the continuous equivalent of the image $[f_{n,m}]$ as

$$f(x,y) = \sum_{n=-N}^{N}\sum_{m=-M}^{M} f_{n,m}\delta(x-n\Delta_x, y-m\Delta_y), \tag{5.5}$$

where $\delta$ is the Dirac function and $\Delta_x$ and $\Delta_y$ are the respective sampling periods in $x$- and $y$-direction. In the following, we will assume $\Delta_x{=}\Delta_y{=}\Delta$.

Similarly, we get the continuous equivalent of the filter template $[w_{n,m}]$:

$$w(x,y) = \sum_{n=-N}^{N}\sum_{m=-M}^{M} w_{n,m}\delta(x-n\Delta, y-m\Delta). \tag{5.6}$$

With the continuous equivalents of $[f_{n,m}]$ and $[w_{n,m}]$, the continuous equivalent of (5.3) becomes

$$g(x,y) = \int_t\int_s w(s,t)\,f(x-s,y-t)\,ds\,dt. \tag{5.7}$$

**From space to frequency: Fourier transformation**

Two-dimensional Fourier transformation is defined as

$$H(u,v) = \mathcal{F}\{h(x,y)\} \equiv \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y)e^{-2\pi\mathrm{i}(xu+yv)}dx\,dy. \tag{5.8}$$

Using the above definition and the fact that

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y)\delta(x-a,y-b)dx\,dy = h(a,b), \tag{5.9}$$

the Fourier transformed image becomes

$$F(u,v) = \mathcal{F}\{f(x,y)\} = \sum_{n=-N}^{N} \sum_{m=-M}^{M} f_{n,m} e^{-2\pi \mathrm{i} \Delta(nu+mv)}. \tag{5.10}$$

The transformed image is a continuous function in the frequency domain. The Fourier transformation $G(u,v)$ of the edge map $g(x,y)$ is calculated similar to the Fourier transformed input image $F(u,v)$. If we write the filter function $W(u,v)$ as the transformation of the template $w_{n,m}$,

$$W(u,v) = \sum_{n=-N}^{N} \sum_{m=-M}^{M} w_{n,m} e^{-2\pi \mathrm{i} \Delta(nu+mv)}, \tag{5.11}$$

then

$$G(u,v) = W(u,v) F(u,v). \tag{5.12}$$

This equation describes the filter operation in the frequency domain. A convolution in the space domain is transformed into a simple multiplication in the frequency domain.

According to (5.11), the filter function $W(u,v)$ in the frequency domain is a summation of weighted exponentials. This facilitates an easy Taylor series expansion.

**Taylor series expansion**

The Taylor series expansion of a function $h(u,v)$ around (0, 0) is defined as

$$h(u,v) = \sum_{r=0}^{\infty} \frac{1}{r!} \sum_{p=0}^{r} \binom{r}{p} u^p v^{r-p} \left. \frac{\partial^r h(u,v)}{\partial u^p \partial v^{r-p}} \right|_{(0,0)}. \tag{5.13}$$

Applying this definition to (5.11), the Taylor series of the filter function $W(u,v)$ in the frequency domain becomes

$$W(u,v) = \sum_n \sum_m w_{n,m} \sum_{r=0}^{\infty} \sum_{p=0}^{r} \frac{u^p v^{r-p}}{p!(r-p)!} (-2\pi \mathrm{i} \Delta)^r n^p m^{r-p}, \tag{5.14}$$

with which the transformed edge map $G(u,v)$ can be written as

$$\begin{aligned} G(u,v) &= \sum_n \sum_m w_{n,m} \sum_{r=0}^{\infty} \sum_{p=0}^{r} \frac{u^p v^{r-p}}{p!(r-p)!} (-2\pi \mathrm{i} \Delta)^r n^i m^{r-p} F(u,v) \\ &= \sum_{r=0}^{\infty} \sum_{p=0}^{r} (-2\pi \mathrm{i} u \Delta)^p (-2\pi \mathrm{i} v \Delta)^{r-p} \frac{F(u,v)}{p!(r-p)!} \sum_n \sum_m w_{n,m} n^p m^{r-p}. \end{aligned} \tag{5.15}$$

**From frequency back to space: inverse Fourier transformation**

Equation (5.15) describes the output edge map, transformed to the frequency domain. If we now want a similar description of the edge map in the space domain, i.e., expansion into gradients of different orders, inverse Fourier transformation is required. This yields a Taylor series equivalent of $g(x,y)$:

$$g(x,y) = \sum_{r=0}^{\infty}\sum_{p=0}^{r} \frac{\partial^r f(x,y)}{\partial x^p \partial y^{r-p}} \frac{(-1)^r}{p!(r-p)!} \sum_n \sum_m w_{n,m} n^p m^{r-p}. \tag{5.16}$$

From (5.16) it can be deduced that the image filter described by $w_{n,m}$ can be regarded as composed of a series of differential operators, with the following continuous analogon:

$$g(x,y) = \sum_{p=0}^{\infty}\sum_{q=0}^{\infty} \alpha_{p,q} \frac{\partial^{p+q}}{\partial x^p \partial y^q} f(x,y), \tag{5.17}$$

with $p+q=r$ and

$$\alpha_{p,q} = \frac{1}{p!q!} \sum_{n=-N}^{N} \sum_{m=-M}^{M} w_{n,m} n^p m^q \tag{5.18}$$

being the coefficients for the gradient vectors for the various values of $p$ (order in $x$-direction) and $q$ (order in $y$-direction).

**Coordinate transform**

For a better insight in the types of differential operators, it can be determined if a filter is directional, and if so, what its main direction of operation is. To this purpose, the $x$- and $y$-axes can be rotated over an angle $\theta$ to new coordinate axes, see Figure 5.3. Now $x$ and $y$ can be written as functions of $\xi$ and $\eta$, depending on $\theta$:

$$\begin{aligned} x &= x_\theta(\xi,\eta) = \xi\cos\theta - \eta\sin\theta \\ y &= y_\theta(\xi,\eta) = \xi\sin\theta + \eta\cos\theta \end{aligned} \tag{5.19}$$

With this transformation, $f(x,y)$ can also be written as

$$f(x,y) = f_\theta(x_\theta(\xi,\eta), y_\theta(\xi,\eta)) = f'_\theta(\xi,\eta). \tag{5.20}$$

N.B. the notation $f'_\theta$ has no relation to the gradient of $f_\theta$.

We can now adapt (5.5)-(5.18) to the coordinate transformation of (5.19). The continuous image function $f'_\theta(\xi,\eta)$ then becomes

$$f'_\theta(\xi,\eta) = \sum_n \sum_m f_{n,m} \delta(\xi - n\Delta\cos\theta - m\Delta\sin\theta, \eta + n\Delta\sin\theta - m\Delta\cos\theta), \tag{5.21}$$

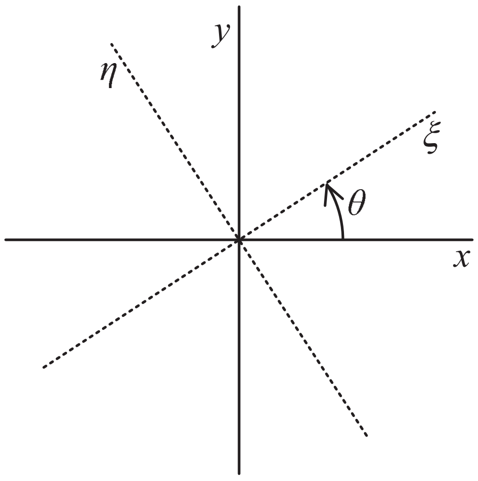


Figure 5.3: Coordinate transformation from $(x, y)$ to $(\xi, \eta)$.

and its Fourier transformed version is then

$$F'_\theta(\mu,\nu) = \sum_n \sum_m f_{n,m} \mathrm{e}^{-2\pi \mathrm{i}\Delta((n\cos\theta + m\sin\theta)\mu + (-n\sin\theta + m\cos\theta)\nu)}. \tag{5.22}$$

In a similar manner as in Section 5.1.5, the output map can be calculated as

$$G'_\theta(\mu,\nu) = W'_\theta(\mu,\nu) F'_\theta(\mu,\nu), \tag{5.23}$$

with

$$W'_\theta(\mu,\nu) = \sum_n \sum_m w_{n,m} \mathrm{e}^{-2\pi \mathrm{i}\Delta((n\cos\theta + m\sin\theta)\mu + (-n\sin\theta + m\cos\theta)\nu)} \tag{5.24}$$

being the Fourier transformed filter function in the transformed coordinate system.

We can now write the Taylor series expansion of the filter function $W'_\theta(\mu,\nu)$ by applying the definition in (5.13) to (5.24):

$$\begin{aligned} W'_\theta(\mu,\nu) &= \sum_n \sum_m w_{n,m} \sum_{r=0}^{\infty} \sum_{p=0}^{r} \frac{\mu^p \nu^{r-p}}{p!(r-p)!} (-2\pi \mathrm{i}\Delta)^r \\ &\quad \cdot (n\cos\theta + m\sin\theta)^p (-n\sin\theta + m\cos\theta)^{r-p} \end{aligned} \tag{5.25}$$

or, using the polynomial equality

$$(a+b)^p = \sum_{k=0}^{p} \binom{p}{k} a^k b^{p-k}, \tag{5.26}$$

we get

$$W'_\theta(\mu,\nu) = \sum_{r=0}^{\infty}\sum_{p=0}^{r}(2\pi\mathrm{i}\mu\Delta)^p(2\pi\mathrm{i}\nu\Delta)^{r-p}\frac{(-1)^{r-p}}{p!(r-p)!}\sum_n\sum_m w_{n,m}$$
$$\sum_{k=0}^{p}\binom{p}{k}(n\cos\theta)^k(m\sin\theta)^{p-k}\sum_{l=0}^{r-p}\binom{r-p}{l}(-n\sin\theta)^l(m\cos\theta)^{r-p-l}. \quad (5.27)$$

Equation (5.27) is again a Taylor series description of the filter function in the frequency domain. Now the complete output edge map can again be calculated in the frequency domain and inverse Fourier transformation results in the following equation in $\xi$ and $\eta$:

$$g'_\theta(\xi,\eta) = \sum_{r=0}^{\infty}\sum_{p=0}^{r}(-1)^r\frac{\partial^r f'_\theta(\xi,\eta)}{\partial\xi^p\partial\eta^{r-p}}\sum_n\sum_m w_{n,m}$$
$$\sum_{k=0}^{p}\sum_{l=0}^{r-p}\frac{(-1)^l n^{k+l}m^{r-k-l}}{k!l!(p-k)!(r-p-l)!}(\sin\theta)^{p-k+l}(\cos\theta)^{k+r-p-l} \quad (5.28)$$

or

$$g'_\theta(\xi,\eta) = \sum_{p=0}^{\infty}\sum_{q=0}^{\infty}\beta_{\theta,p,q}\frac{\partial^{p+q}}{\partial\xi^p\partial\eta^q}f'_\theta(\xi,\eta), \quad (5.29)$$

with $p+q=r$ and

$$\beta_{\theta,p,q} = (-1)^{p+q}\sum_{n=-N}^{N}\sum_{m=-M}^{M}w_{n,m}$$
$$\sum_{k=0}^{p}\sum_{l=0}^{q}\frac{(-1)^l n^{k+l}m^{p+q-k-l}}{k!l!(p-k)!(q-l)!}(\sin\theta)^{p-k+l}(\cos\theta)^{q+k-l}. \quad (5.30)$$

Equation (5.30) gives the Taylor series coefficients of the edge detection filter template, from which we can deduce of which orders of differential operators the filter consists, i.e., those $p$ and $q$ that give the larger $\beta_{\theta,p,q}$, and in which direction(s) these operators work optimally, i.e., the angle(s) $\theta$ for which $\beta_{\theta,p,q}$ is maximal for certain $p$ and $q$. This can be represented graphically by drawing the value of $\beta_{\theta,p,q}$ as a function of $\theta$ for various $p$ and $q$. See Figures 5.9 and 5.12 in Section 5.2.2 (Results) for some examples. In these graphs, the normalized absolute value of $\beta_{\theta,p,q}$ as a function of $\theta$ is represented by the distance from the center of the graph in the direction of $\theta$; positive values of $\beta_{\theta,p,q}$ are shown as thick dark lines, negative values as thin lines. More detailed results are presented in the next section.

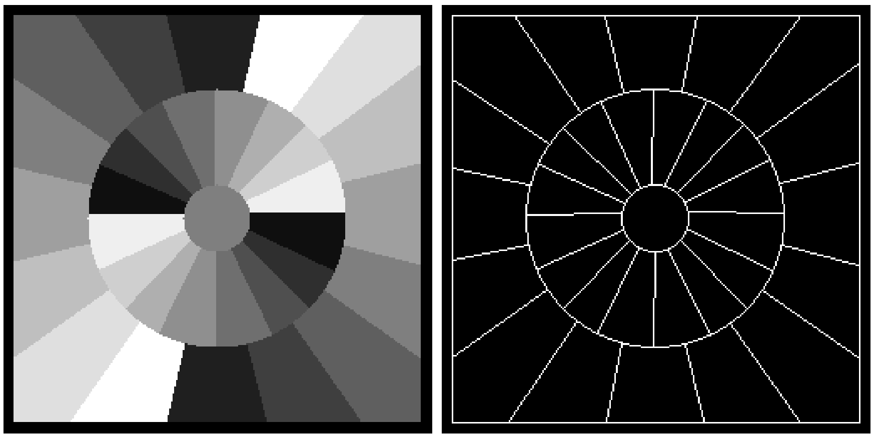


Figure 5.4: A $128 \times 128$-pixel training image (*left*) and corresponding reference edge map (*right*).

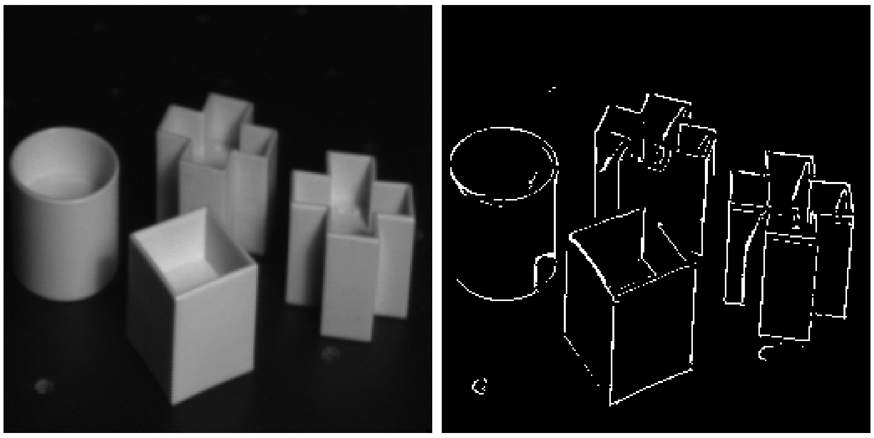


Figure 5.5: Test image (*left*) and thresholded result after edge detection by a neural network (*right*).

## 5.2 Analysis of image processing systems

This section presents the results obtained from an analysis of several pixel classificators, some of which were realized with feedforward neural networks, in terms of the generic domain-dependent basic functions identified and introduced in the previous sections.

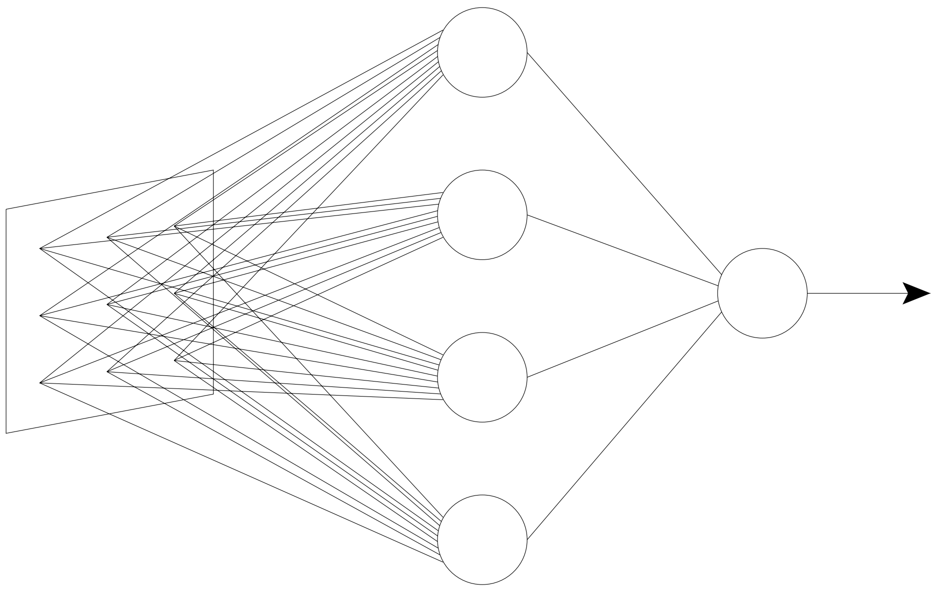


Figure 5.6: A neural network with 9 inputs, 4 hidden neurons, and one output used for edge detection.

### 5.2.1 Neural network edge detector

In order to test our analysis method, we trained several artificial neural networks for edge detection. The neural networks were of the feedforward–error-backpropagation type, with $3 \times 3$ to $11 \times 11$ inputs, 4 to 8 units in the single hidden layer, and a single output. See Figure 5.6 for an example. All units used sigmoid activation functions. Some networks were trained with a training image containing sharp edges only (see Figure 5.4). A different image was used as a test set, see Figure 5.5 for a test result by a neural network edge detector. Other networks were trained with a similar image as the one in Figure 5.4, but containing sharp edges as well as blurred ones, and that had Gaussian noise added to one half of the image. This made the neural network edge detectors less sensitive to noise, as also becomes visible in the analysis results shown hereafter in Section 5.2.2.

### 5.2.2 Results

As (5.30) gives the Taylor series coefficients of any image filter template, we can apply it to existing edge detector templates as well as to a neural network edge detector's hidden units, whose weight matrices can also be interpreted as templates.

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

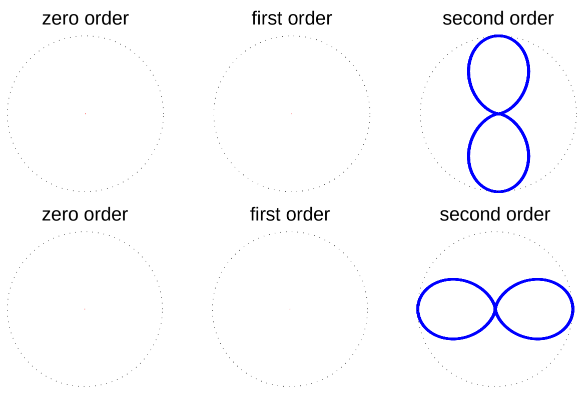


Figure 5.7: Templates and low-pass, gradient, and second-order gradient analysis results for a horizontal (*top*) and a vertical (*bottom*) line detector.

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

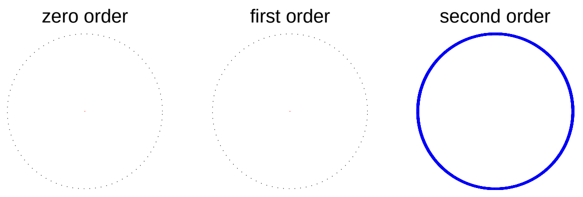


Figure 5.8: Template and low-pass, gradient, and second-order gradient analysis results for a spot detector.

$$\begin{bmatrix} -5 & -5 & 3 \\ -5 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

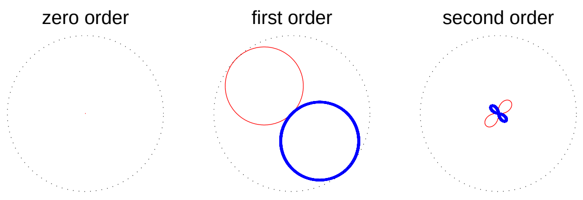
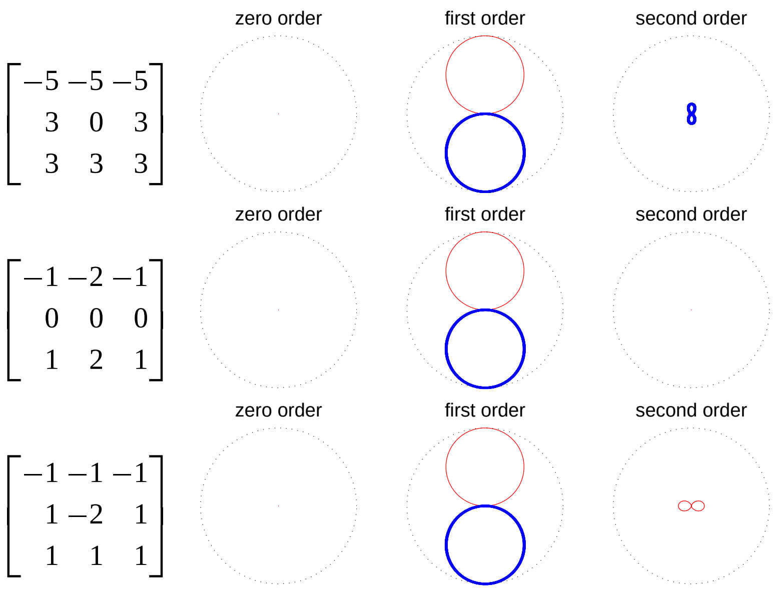


Figure 5.9: Template and low-pass, gradient, and second-order gradient analysis results for a diagonal Kirsch edge detector template.

First, we give brief analysis results for some of the line and spot detector templates shown in Sections 5.1.2 and 5.1.3. In the case of line-detecting templates, it is clear from the first template shown in Section 5.1.2, that it detects horizontal lines, and from the second one that it detects vertical lines, therefore it is not surprising that the second-order coefficients are strongest in the vertical and horizontal directions respectively, as seen in Figure 5.7. The zero-order and first-order coefficients are all zero. This is also the case for the spot detector, see Figure 5.8, except that in this case the second-order behavior is indeed omnidirectional, or rotation-invariant, as already indicated in Section 5.1.3.



$$\begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 5.10: Templates and low-pass, gradient, and second-order gradient analysis results for three horizontal edge detector templates: Kirsch (*top*), Sobel (*middle*), Prewitt compass (*bottom*).

Next, we show analysis results for the diagonal Kirsch template shown in Section 5.1.1. As can clearly be seen from Figure 5.9, this template shows no low-pass (averaging) behavior, has a strong first-order gradient operation in diagonal direction, as could be expected, and weak second-order gradient behavior. A similar result is seen in Figure 5.10, where a horizontal Kirsch template is compared with two other horizontal edge detection templates. The differences between these three are only visible in their second- (and higher-) order behavior.

Figure 5.11 shows results for the 4 hidden units of a small neural network edge detector, which was trained with the sharp edges shown in Figure 5.4. For the purpose of showing the hidden units' weight values graphically, they have been scaled to values between –1 (black) and +1 (white). The weight templates shown in the first column of Figure 5.11 already give some insight into their behavior, but the Taylor series coefficient analysis clearly shows that three of the units detect edges in various directions, and one unit acts as a second-order gradient filter with minor first-order gradient behavior. Notice that all four units do not have a significant low-pass (zero-order gradient) component.

Another neural network with the same architecture was trained with sharp, blurred, and noisy variants of the images shown in Figure 5.4. The weight templates of the hidden units are shown in Figure 5.12 along with the graphical representations of

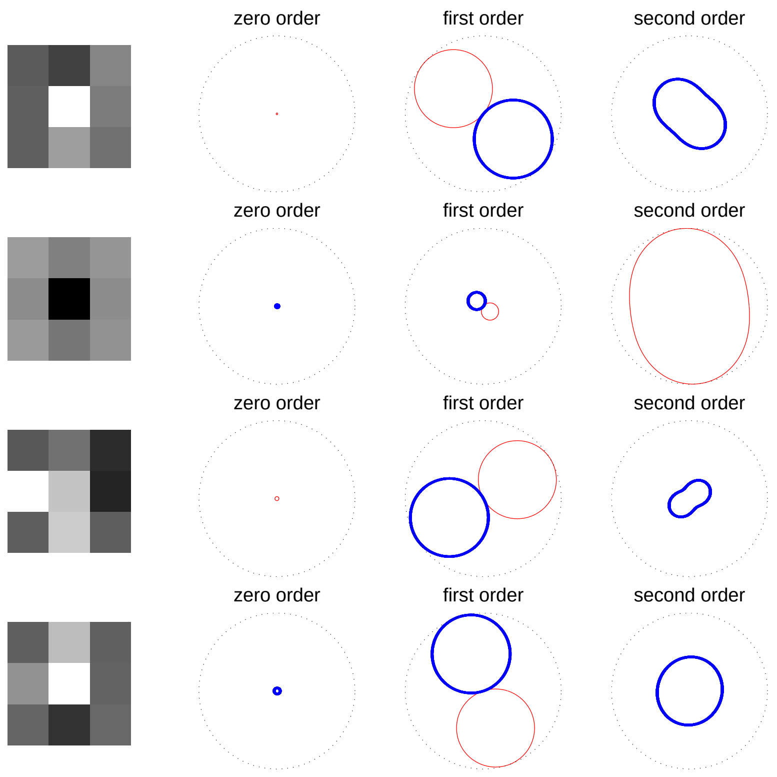


Figure 5.11: Weight templates and low-pass, gradient, and second-order gradient analysis results for all 4 hidden units of a neural network edge detector with $(3 \times 3)$ inputs, 4 hidden units, and one output unit, trained with sharp edges.

their Taylor series coefficients. This network's units have similar gradient components as the previous one, although the second-order gradient components are somewhat stronger. More remarkably, however, is the fact that the low-pass components are significantly present, as compared to the previous network. This can be explained by the differences in the training sets with which the networks were trained. Low-pass or averaging behavior makes the network less sensitive to noise and improves the edge detection ability of the second network.

Although the above only gives some analysis results for the units in the hidden layer, it should be clear that a description of the neural network as a whole can be derived from these results. The weight between a hidden units and the output unit represents the "importance" of the hidden unit's edge detection outcome, which is then combined with the other hidden units' outcomes into a single answer indicating whether the pixel under investigation belongs to an edge or not.

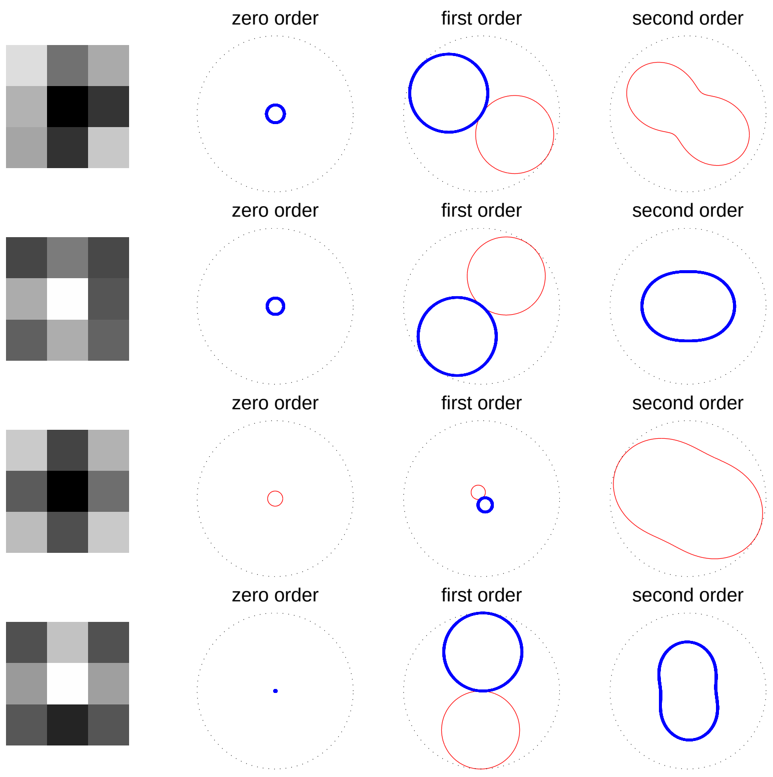


Figure 5.12: Weight templates and low-pass, gradient, and second-order gradient analysis results for all 4 hidden units of a neural network edge detector with $(3 \times 3)$ inputs, 4 hidden units, and one output unit, trained with sharp/blurry/noisy edges.

Some larger neural networks have also been trained and analyzed, with similar results, although in general, the larger the network, the more variety in behavior among the neural units. In a few cases, certain units showed very strong higher-order behavior, indicating that those units functioned as noise detectors only. Although such units usually have weak connections to the output unit (low importance), removing them from the network (pruning) often results in worse edge detection capabilities for the network as a whole. This is because the noise detecting unit decreases the confidence of an edge detection outcome if the local neighborhood around the pixel under investigation is very noisy. Units that are specialized in the detection of sharp edges could easily misclassify such pixels as edge pixels.

# Chapter 6

# A case study in classification

This chapter illustrates the analysis of a feedforward neural network that has been trained for a classification task. First, the neural network-based classification system will briefly be introduced. Then, the knowledge that is stored in the trained neural network will be extracted and visualized in the form of class prototypes. In the application area of classification systems, class prototypes can be considered as basic functions, as they ideally represent the basic features of the classes of patterns that are present in the input space. This chapter is mainly based on an earlier publication (Van der Zwaag et al. 2003d), with some recent additional results.

## 6.1 Classification in general

In this section we describe the analysis of feedforward neural networks in the application field of classification problems, by translating the trained network into a feature map such as used in Kohonen's self-organizing feature maps (SOMs) (Kohonen 1982). The structural composition by a set of basic functions that have a known translation into the functionality of a SOM, allows transforming the overall structure into a feature map. The gained benefit is the transparency of the stored knowledge from the resulting SOM.

In general, artificial neural networks with unsupervised training merely reorganize the input space, so analyzing them after training becomes fairly simple: an investigation into the reorganized input space reveals how the network has restructured the input space.

Analyzing neural networks trained under supervision is far more complicated, for input and output spaces are usually in different domains (e.g., a character recognition system has an image as input, and a character class as output), whereas in the unsupervised case, input and output spaces are basically the same, although they are organized in different ways.

Generic operators to be used as basic functions such as in the previous chapter can be readily identified in a SOM as well. This significantly improves the comprehension of the knowledge that is stored in the networks. In the general domain of classification problems, the mapping of the classes in the input space can be regarded as a suitable basic function. Such a mapping is characteristic for self-organizing maps, so if one translates a feedforward network into a SOM-like network, a comprehensible description of the neural network's functionality is easily accomplished. Though the transformation is not straightforward, it holds the promise to be computable in limited time, where the alternative of straight rule extraction falls short.

In the general domain of classification problems, the prototyping of the classes in the input space can be regarded as another suitable basic function. Such a prototyping is characteristic for self-organizing maps, where prototypes are mapped onto a topology-preserving feature map. So, if one could determine accurate prototypes for the classification realized by a feed-forward neural network, again a comprehensible description of the neural network's functionality would be accomplished. In the following sections we will describe this method in more detail.

## 6.2 Determining prototypes

One of the graphical representation methods occasionally seen in literature is basically a graphical representation of the synaptic weights of the network, in particular those between the input and hidden layers. See Section 6.4 for an example. This is usually based on the idea that class features are stored in the network's synaptic weights and that these features can be identified by direct visual inspection of the weights, ordered in the same way as the input vectors.

Unfortunately, the visualization of the synaptic weights in that way does not give insight into the knowledge stored in the network. Therefore, we propose to determine feature vectors, or prototypes, from the neural network. In this sense, we analyze the neural network in terms of domain-dependent basic functions, which in this case we choose to be class prototypes.

Whereas determining prototypes from a Kohonen self-organizing map is easily achieved by directly reading the feature vectors stored in the neurons, this is not straightforward when dealing with feedforward networks. We will now describe the method we used to determine prototypes in two main steps.

### 6.2.1 Estimating internal prototypes

Suppose we want to determine a prototype for class $C$. Suppose that the target vector used for teaching examples of class $C$ is $\mathbf{t}^C$. We know that the ideal class $C$ output of the network is then

$$\mathbf{y} = \mathbf{t}^C. \tag{6.1}$$

From Section 2.3.1, we also know that the network output is given by

$$\mathbf{y} = f(\mathbf{W}_K \mathbf{o}_{K-1}) \quad \text{or} \quad y_i = f(\sum_j w_{Kij} o_{K-1,j}) \quad \text{for all } i, \tag{6.2}$$

where $f(x)$ is the activation function of the output neurons.

In order to ensure ideal classification of the prototype for class $C$, we must maximize $f(\sum_j w_{Kij} o_{K-1,j})$ for all $i$ for which $t_i^C$ is high and minimize the same for all $i$ for which $t_i^C$ is low. With any activation function like one of those shown in Chapter 2, this would mean that we are looking for $\sum_j w_{Kij} o_{K-1,j} \rightarrow \infty$ and $\sum_j w_{Kij} o_{K-1,j} \rightarrow -\infty$, respectively. This is not realizable, as the $o_{K-1,j}$ are bounded between 0 and 1, or between $-1$ and 1, depending on the chosen activation function.

However, what we can do is define for layer $k$ an internal prototype $\mathbf{p}_k^C$ for class $C$. For the output layer $K$ we define the output prototype as

$$\mathbf{p}_K^C = \mathbf{t}^C, \tag{6.3}$$

the target vector for class $C$.

The hidden layer internal prototypes are defined as

$$p_{ki}^C = \begin{cases} 1 & \text{if } \sum_j w_{k+1,j,i} p_{k+1,j}^C > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6.4}$$

These hidden layer output prototypes will ideally yield a maximum network output for output class $C$ and minimum output for all other classes.

### 6.2.2 Estimating class prototypes

The next step is to name the internal prototype $\mathbf{p}_k^C$ the ideal $k$th layer internal output for class $C$. Preliminary experiments have led to the following recipe for using this idealized output to estimating the $(k-1)$th layer internal prototype $\mathbf{p}_{k-1}^C$ for class $C$:

$$\mathbf{p}_{k-1}^C = (N_k - \Sigma \mathbf{p}_k^C)\mathbf{p}_k^C \mathbf{s}_{\mathbf{W}_k} - \Sigma \mathbf{p}_k^C (1 - \mathbf{p}_k^C)\mathbf{s}_{\mathbf{W}_k}, \tag{6.5}$$

where $N_k$ is the number of neurons in layer $k$ and $s_{\mathbf{w}_{ki}}$ is a Boolean variable with value 1 for those $j$ for which $w_{kij} < 0$ and with value 0 for all other $j$.

One should realize that it is generally impossible to determine *the* prototype for a given class, as the mapping that is realized by the neural network is not a one-to-one mapping. In other words, different input patterns can yield the same network output. Therefor, in theory it is not possible to invert this many-to-one mapping in order to get unique prototypes. However, we expect that the different input patterns that lead to a given output class can be sufficiently represented by well-estimated prototypes.

In order to test the accuracy of the prototypes that are found with the method described above, the final class prototypes can be offered to the inputs of the network. If the prototypes are indeed representative for their classes, the network output will be equal to the target output for the respective classes.

## 6.3 Classification with a feedforward neural network

In order to illustrate the above-described analysis method applied to a neural network classifier, we have trained a feedforward neural network for the classification of a set of characters from the Latin alphabet. The set that we used is available from the Matlab[1] Neural Network Toolbox. It is shown in Figure 6.1.

The feedforward back-propagation network that we used had 35 inputs, fully connected to one hidden layer containing 10 hidden neurons, which were again fully connected to the output layer holding 26 output neurons (Figure 6.2). The neurons in the hidden and output layers all used the logarithmic sigmoid shown earlier in Figure 2.3 as their activation function. During training, clean as well as noisy images were offered to the network inputs, and for each character image, the target vector for the output layer contained a one for the output neuron corresponding to the position of the character in the alphabet and zeros otherwise. The noisy images were created by adding uniformly-spread noise of up to 0.2 to the values of the input pixels (original values either 0 or 1). After training, all training patterns were correctly classified by the network. Nevertheless, for our purpose, i.e., illustrating knowledge extraction from the trained network, it is relatively unimportant how well the network classifies the learned patterns, or how sensitive it is for different levels of noise.

[1] © The Mathworks, Inc.

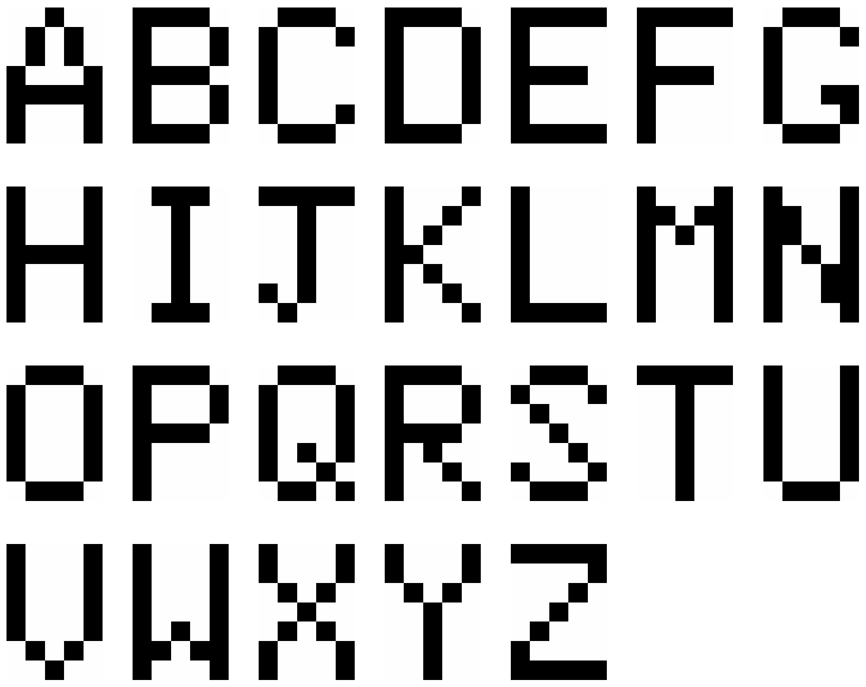


Figure 6.1: A set of twenty-six $5 \times 7$-pixel characters.

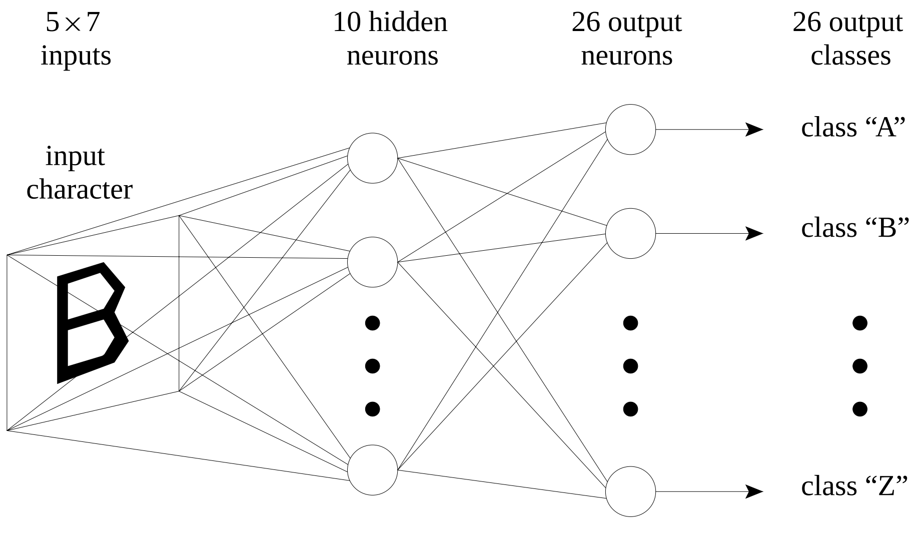


Figure 6.2: A feedforward network used for classification of character images.

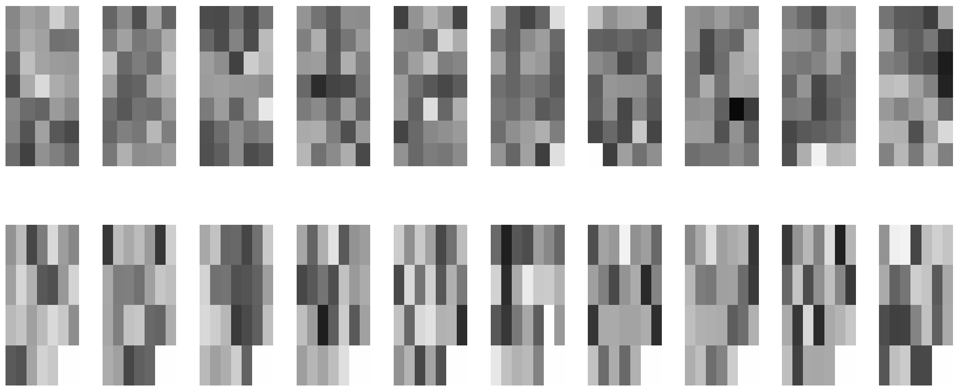


Figure 6.3: Graphical representation of the synaptic weights into (*top*) and from (*bottom*) the hidden nodes.

## 6.4 Knowledge representation

One of the graphical representation methods occasionally seen in literature is basically a graphical representation of the synaptic weights of the network, in particular those between the input and hidden layers. We can also do this for our network, resulting in the "feature map" presented in Figure 6.3. This figure shows the synaptic weights (35 per neuron) in matrix form for all ten hidden neurons, as well as the synaptic weights (26 per neuron) between the hidden and output layers, organized per hidden unit.

Unfortunately, the visualization of the synaptic weights in the way shown in Figure 6.3 does not give insight into the knowledge stored in the network. Therefore, we use the analysis method described in Section 6.1 to estimate feature vectors, or prototypes, from the neural network. In this sense, we analyze the neural network in terms of domain-dependent basic functions, which in this case we choose to be class prototypes.

## 6.5 Results

If we apply the method described in Section 6.1 to all classes of our feedforward network, we get the resulting prototypes shown in Figure 6.4. This figure shows the synaptic weights, normalized to values between 0 and 1 and mapped to grayscale values for visualization purposes. Even though the similarities with the original training patterns are weak (yet still visible, e.g., at classes "K", "O", "U" or "X"), the network classifies 24 out of the 26 prototypes correctly (most of them with

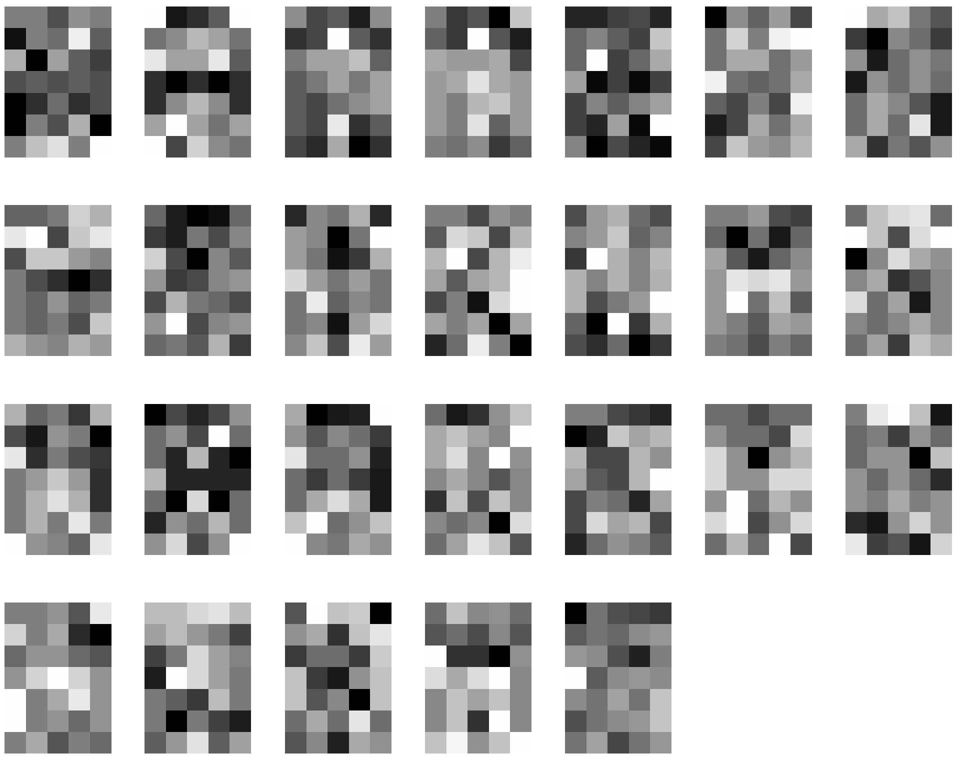


Figure 6.4: Prototypes of all 26 classes.

confidence over 90%) when offered to the network. The misclassifications occur with the derived prototype of a “B”, which is misclassified as a “Q” and the derived prototype of a “W”, which is misclassified as a “G”.

The inaccuracy of the derived prototypes may partially be caused by the noninvertability of the network function due to the bounds on the neuron outputs. Another possible cause is the noninvertability due to the many-to-one mapping that the network realizes. However, for a small network such as the one in our experiments, a more accurate internal prototyping can be achieved by calculating the network output over the whole range of possible hidden layer outputs. Even then, the number of possible hidden layer outputs is infinite, because of the continuous activation function. Therefore, we assume the hidden layer outputs to be equal to either 1 or 0. For our network with 10 hidden neurons, this means an evaluation of $2^{10} = 1024$ cases, which is still feasible. The internal prototypes can then be determined by choosing the hidden layer outputs with the highest distinction in the network output, i.e., for which the difference between the output value for the target class and the highest output value for any other class is maximal.

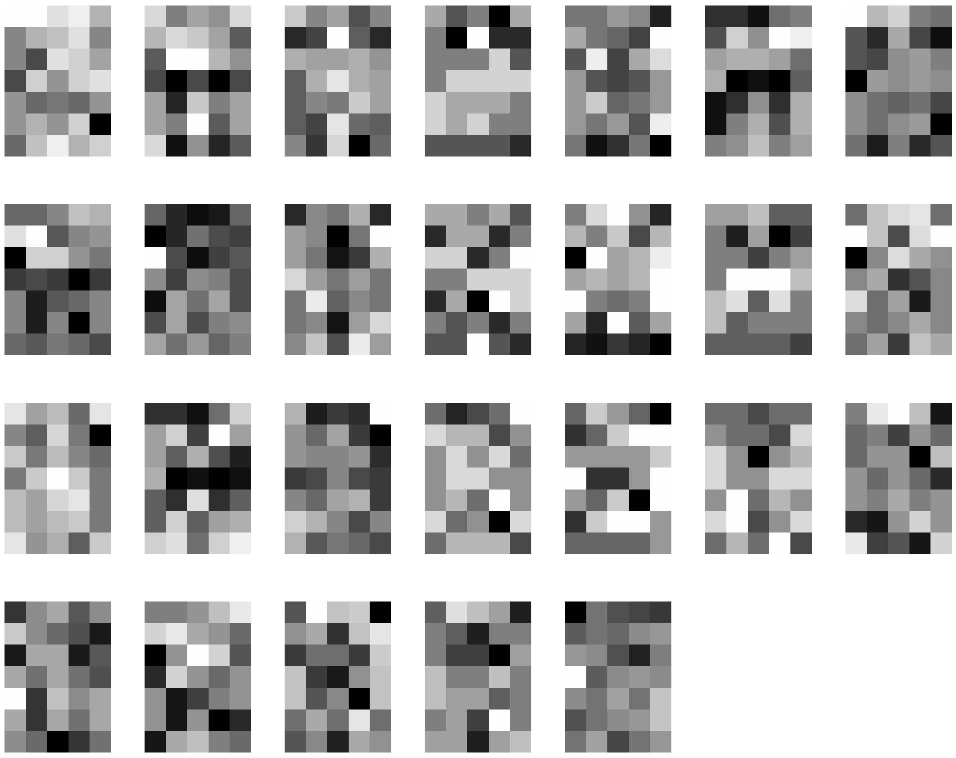


Figure 6.5: Improved prototypes of all 26 classes.

The new internal prototypes can then again be used for the derivation of the final prototypes, using (6.5). The graphical result, shown in Figure 6.5, is similar to that of Figure 6.4, but the network performance when the new prototypes are offered to the neural network has improved. All 26 prototypes are now correctly classified by the network. It is expected that more improvements in our prototype method are possible. This will require more research.

As a way of testing the prototype estimation methods, we estimated class prototypes for another network that was trained for the same task as the network discussed above. However, the new net had 10,000 hidden neurons instead of just 10. Although this is quite an overkill for the relatively small classification problem with which we are dealing here, we can expect that the stored knowledge will be spread over many more network parameters in this enlarged network. In other words, the information stored in one parameter in the small network can be spread over a large number of parameters in the large network. This benefits the estimation of prototypes, as small estimation errors due to one network parameter are expected be less prominently visible in the resulting prototypes, if the errors are partially compensated by similar information stored in other parameters in the large network.

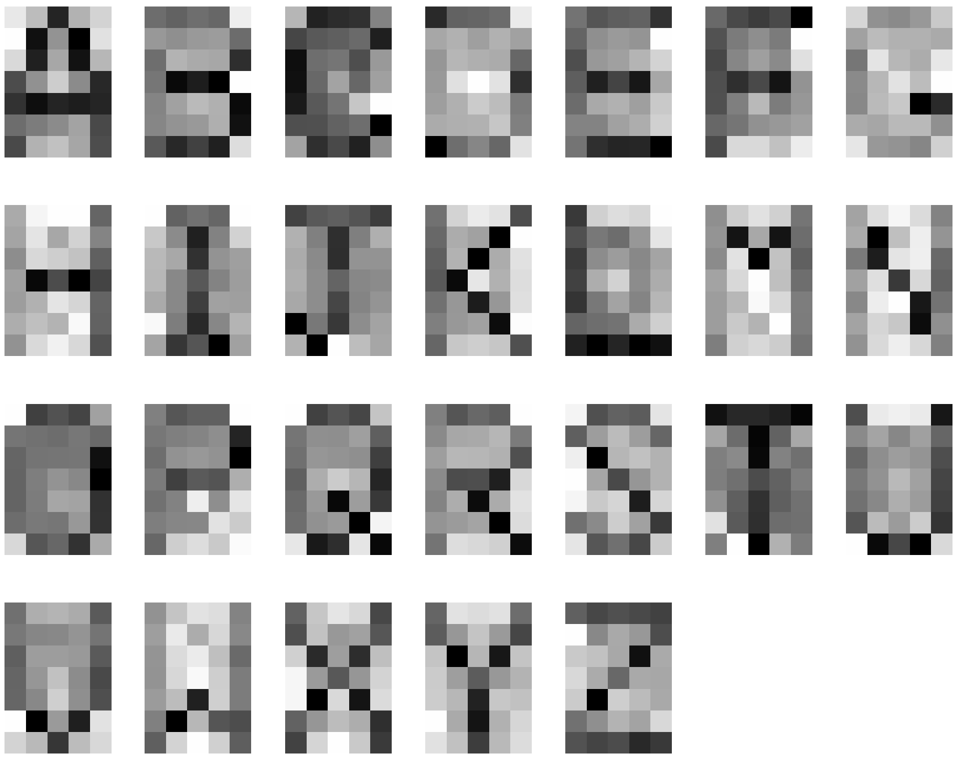


Figure 6.6: Prototypes of all 26 classes for a large network.

Of course, in this case the last-mentioned method for estimating the internal prototypes, i.e., by an “exhaustive” search, is not feasible, as it would involve evaluating the network output for $2^{10000} \approx 10^{3000}$ cases. For this reason, the estimation method mentioned earlier was used again. Contrary to the prototype testing result in the case of the network with 10 hidden neurons, this method yielded 100% correctly classified prototypes in the case of the network with 10,000 hidden neurons.

As is clearly visible from the results shown in Figure 6.6, the estimated prototypes for the large network are indeed more similar to the original input patterns. This improvement can also be quantified. The average squared error between the original clean input patterns and the estimated prototypes shown in Figure 6.5 is 0.19 on a scale of 0 to 1, and the average squared error between the original clean input patterns and the estimated prototypes shown in Figure 6.6 is only 0.09.

An interesting aspect can be found if the estimated prototypes are inspected more closely. From the found class prototypes we can derive the importance of certain input elements (pixels) for a given class. For instance, a comparison of the prototypes for classes “C” and “G” shows the importance of the two pixels that form the difference between these two characters. White and black pixels in the prototypes can be considered to be important, and gray pixels as less important. Other examples of this feature can be seen in a comparison of the prototypes for class “D” with those of classes “B” and “O”. A practical application of this feature could be in neural networks for face recognition, so as to find out which parts of the face are considered important by the neural network, or which parts of the face might cause a failed recognition, if the network considers those parts unimportant.

## 6.6 Discussion

We have trained neural networks to classify characters and analyzed them in terms of class prototypes. From the results displayed and described in the previous sections it is clear that it is indeed feasible to describe the trained neural networks in terms of basic functions from the classification domain, although improvements can still be expected.

For example, one important aspect of self-organizing maps is their topology preserving character. In the map of prototypes that we have derived in Figures 6.4 to 6.6, no topological information is present. That is, the class prototypes are just shown in their alphabetical order, without any present indication of closer similarities between neighboring classes than between classes shown further apart from each other in the map of prototypes.

Nevertheless, the description with class prototypes gives an impression of the class knowledge stored in the synaptic weights of the neural network as a classifier, even though the prototypes seem not very likely as input patterns for this particular classification problem if the neural network is relatively small. A tempting conclusion would be that the term *ideal prototype* is interpreted differently by humans and artificial neural networks.

To improve the algorithms for the extraction of prototypes from trained feedforward neural network classifiers, and to enhance the interpretation of the derived prototypes, we recommend a deeper investigation. Other explorations could also be recommended, such as the investigating the possibility of translating self-organizing maps into feedforward networks, for example for hardware considerations.

Chapter 7

# Conclusions

This thesis presents a novel generic method for the analysis of trained feedforward neural networks. The analysis is based on a description of (parts of) the trained neural networks in terms of domain-dependent basic functions. Suggestions for suitable basic functions are presented for a number of application domains. For the digital image processing domain and for the general field of classification problems, case studies are presented. Neural networks that had been trained for specific tasks within those domains have been analyzed to illustrate some of the possibilities of the methods described in this thesis.

For example, we have trained neural networks to detect edges in digital images and analyzed them into gradient filter components. From the results displayed and described in Section 5.2 it is clear that it is indeed feasible to describe the trained neural networks in terms of basic functions from the image processing domain. Even non-neural templates from classical image processing literature have been successfully analyzed.

The description with gradient filter components gives easy insight into the behavior of the neural network as an edge detector, and allows simple comparison with other edge detectors which can in the same way be described in terms of gradient filter components.

For the application domain of classification systems we have trained a feedforward neural network to classify characters and analyzed the network in terms of class prototypes. From the results displayed and described in Section 6.3 it is clear that it is indeed feasible to describe the trained neural network in terms of basic functions from the classification domain, although improvements can still be expected.

For example, one important aspect of self-organizing maps is their topology-preserving character. In the map of prototypes that we have derived in Figures 6.4 to 6.6, no topological information is present. That is, the class prototypes are just shown in their alphabetical order, without any present indication of closer similarities between feature vectors from neighboring classes than between those from classes shown further apart from each other in the map of prototypes.

Nevertheless, the description with class prototypes gives an impression of the class knowledge stored in the synaptic weights of the neural network as a classifier, even though the prototypes seem not very likely as input patterns for the particular classification problem displayed in Section 6.3 if a small network is used.

In general, the analysis consists in describing the internal functionality of the neural network in terms of domain-dependent basic functions, functions that can be considered basic in the application domain of the neural network. This means that users who may not be familiar with artificial neural networks, but who are familiar with basic functions that are often used in their problem domain, can gain insight in the way the neural network solves their problem. For such users, this is often an important factor in deciding to apply artificial neural networks to a problem that may be difficult to solve otherwise.

The novel analysis method introduced in this thesis offers a much wider applicability than existing methods, some of which can be defined as specific cases of our generic method.

The analysis method presented here is not only very suitable for visualization of the knowledge that is present in a trained neural network, but it offers several new prospects to the users of those neural networks. The domain knowledge that is extracted from the neural network can be regarded as a set of knowledge modules. Once users are presented with extracted domain knowledge in the form of identifiable modules, they can benefit from this by feeding the modular knowledge back into the system in such a way that the system is enhanced and rebuilt in a more structural way.

Identifiable knowledge blocks fit very well in the 3-tier model in which basic knowledge is present in a modular way in the bottom layer. A system that has a modular 3-tier structure, can in turn be much more easily analyzed, as the expert knowledge that is present in the system can be easily located and extracted.

## 7.1 Recommendations

In order to have a better overview of the possibilities of the generic method for the analysis of artificial neural networks, it is recommended that in addition to the application fields included in this thesis, more application domains are investigated. Some of these domains will require very specific domain expert knowledge in order to be able to select a suitable set of generic domain-specific basic functions. Hence, neural network experts should work in close cooperation with experts in those application areas.

For the general field of classification problems, it would be desirable to enhance the method that was elaborated in Chapter 6, in order to be able to say more about the generalizability of the extracted sets of prototypes. Currently, it is unclear whether the extracted prototype patterns lie in the "middle" of the classes they represent, or closer to the boundary regions that lie between the classes. This could, for example, be checked by feeding sets of test patterns to a constructed map of extracted prototypes, and testing whether the test patterns indeed belong to the class represented by the prototype that has the smallest difference with the respective test pattern (cf. the winning neuron in normal self-organized maps).

As for the case study in the image processing domain, it would be useful to have an algorithm that calculates a set of gradient filters that represents the network as a whole. This will typically be a nonlinear combination of the gradient filters extracted from the hidden neurons, with respect to the activation functions and the synaptic weights of the output layer. Such an "overview collection" or "summary set" of gradient filters would be particularly useful for the analysis of large networks.

# Appendix A

# Results of a patent search

In order to get an impression of the relative interest in the analysis of artificial neural networks as compared to the total spectrum of artificial neural networks, it is useful to have a look at patents that deal with either neural networks or with neural networks and their analysis. A search in several patents databases results in a conclusion that around 20% of the patents dealing with neural networks mention both neural networks and some form of analysis (e.g., rule extraction). Within these 20%, approximately 80% are actually treating analysis systems using neural networks rather than analysis of neural networks themselves, which forms only 2.5% of all neural network-related patents. Within this remaining fraction, most finds were caused by cited literature references or other coincidental co-appearances of the search terms. The number of possibly interesting patents is no more than 36 (see Table A.1), which drops to just 15 more or less relevant patents when patents appearing more than once within and between databases are only counted once.

## A.1 Sensitivity analysis

Four patents describe the use of input-output sensitivity analysis of neural networks, or "internal causality analysis":

> Enbutsu, I., Baba, K., Watanabe, S., Yahagi, H., Hara, N., and Yoda, M.: "Plant operation support system," US 005 581 459 A, granted 3 Dec. 1996.
>
> Keeler, J.D. and Hartman, E.J.: "Method and apparatus for analyzing a neural network within desired operating parameter constraints," US

Table A.1: Relevant results of patent searches in the following databases: United States Patents from 1971 onwards (US), European Patent Applications (EPA) and Specifications (EPB), Patent Abstracts of Japan (JP), and World Intellectual Property Organization (WO). The searches were performed on 16 May 2001.

| database | total no. of patents | no. of patents related to NNs | idem, relevant to NN analysis | idem, without duplicates |
|---|---|---|---|---|
| US | 2,815,187 | 2838 | 10 | 6 |
| EPA | 1,320,485 | 853 | 6 | 3 |
| EPB | 517,525 | 239 | 1 | 1 |
| JP | 5,599,037 | 3166 | 16 | 12 |
| WO | 553,047 | 470 | 3 | 2 |
| total | 10,805,281 | 7566 | 36 | **15** |

005 781 432 A, granted 14 July 1998.

Keeler, J.D., Hartman, E.J., and Liano, K.: "Method and apparatus for determining the sensitivity of inputs to a neural network on output parameters," US 005 825 646 A, granted 20 Oct. 1998.

Tsuzuki, H.: "Method for analysis processing neural network," JP 06 052 133 A2 (English language abstract), application date 29 July 1992.

## A.2 Fuzzy rule extraction

Four patents describe the use of fuzzy rule extraction from neural networks:

Abe, S., Volkmar, U., and Ming, S.R.: "Fuzzy rule generating method and its device, fuzzy inference method and its device," JP 07 028 767 A2 (English language abstract), application date 13 July 1993.

Inoue, T.: "Fuzzy-neural network system," US 005 740 322 A, granted 14 April 1998.

Khan, E.R.: "Fuzzy logic design generator using a neural network to generate fuzzy logic rules and membership functions for use in intelligent systems," US 005 579 439 A, granted 26 Nov. 1996.

Levey, C.A.: "Neural network having expert system functionality," US 005 398 300 A, granted 14 March 1995.

## A.3 Boolean rule extraction

Three patents describe the use of Boolean rule extraction from neural networks:

> Niida, K., Koshijima, I., Tani, A., and Hirobe, T.: "Causal relation rule extraction method by neural network," JP 04 064 163 A2 (English language abstract), application date 3 July 1990.
>
> Saito, K.: "Method and device for rule extraction from neural network," JP 07 234 852 A2 (English language abstract), application date 24 Feb. 1994.
>
> Tsukimoto, H. and Morita, C.: "Method and device for analyzing neural network," JP 09 190 419 A2 (English language abstract), application date 12 Jan. 1996.

## A.4 Other methods

One patent describes the use of geometric interpretation of hidden layer units in feedforward neural networks (see also (Mitchell 1992)):

> Mitchell, J.: "Optimisation of feedforward neural networks," EP 0 583 217 B1, granted 10 May 2000.

One patent describes the pruning of hidden units by determining their independency:

> Tamura, S.: "Method for structuring feedforward type neural network," JP 07 230 437 A2 (English language abstract), application date 21 Feb. 1994.

One patent describes a method of analyzing the data content of a neural network by textual labeling of clusters in the input data space. From the abstract it is not clear which type of neural network is meant (the type is essential, as such methods are inherent to certain neural network types, for instance Kohonen's self-organizing feature map (Kohonen 1982)):

> Narasaki, H., Watanabe, T., Konishi, M., Yamamoto, M., Asada, H., and Hamaguchi M.: “Holding data analysis method for neural network,” JP 07 029 006 A2 (English language abstract), application date 7 July 1993.

One patent describes the use of linear approximation of hidden units, which by combination results in an estimated input-output relation for the entire neural network. This is actually an interesting approach, as neural networks themselves are usually applied to estimate (nonlinear) input-output relationships of other systems.

> Kimoto, T. and Asakawa, K.: “Function analyzing device for learning device,” JP 03 118 658 A2 (English language abstract), application date 30 Sep. 1989.

# References

Anderson, J.A., Silverstein, J.W., Ritz, S.A., and Jones, R.S. (1977), “Distinctive features, categorical perception, and probability learning: some applications of a neural model,” *Psychological Review*, vol. 84, pp. 413-451. Reprinted in Anderson, J.A. and Rosenfeld, E. (eds.) (1988), *Neurocomputing: Foundatons of Research*, Cambridge, MA: The MIT Press.

Andrews, R., Diederich, J., and Tickle, A.B. (1995), “A survey and critique of techniques for extracting rules from trained artificial neural networks,” Neurocomputing Research Centre report, Queensland University of Technology, Australia.

Auda, G. and Kamel, M. (1999), “Modular neural networks: a survey,” *Int. Journal of Neural Systems*, vol. 9, no. 2, pp. 129-151.

Barakova, E. (1999), *Learning Reliability: a Study on Indecisiveness in Sample Selection*, Ph.D. thesis, Rijksuniversiteit Groningen, the Netherlands.

Blake, C.L. and Merz, C.J. (1998), “UCI repository of machine learning databases,” University of California, Dept. of Information and Computer Science, Irvine, CA. Available at http://www.ics.uci.edu/˜mlearn/MLRepository.html .

Blattner, J., Neuer, S., Spaanenburg, L., and Nijhuis, J.A.G. (1993), “Optimizing fuzzy rules by neural learning,” in: *Digest International Conference on Mathematical and Intelligent Models in System Simulation MISS’93* (Brussels), pp. 238-246.

Bolt, G. (1991), “Investigating Fault Tolerance in Artificial Neural Networks,” Technical Report no. YCS 154, Department of Computer Science, University of York, U.K.

Carpenter, G.A. and Grossberg, S. (1987), “A massively parallel architecture for a self-organizing neural pattern recognition machine,” *Computer Vision, Graphics and Image Processing*, vol. 37.

Choi, J.Y. and Choi, C.-H. (1992), “Sensitivity analysis of multilayer perceptron with differential activation functions,” *IEEE Transactions on Neural Networks*, vol. 3, no. 1, pp. 101-107.

Craven, M.W. and Shavlik, J.W. (1994), “Using sampling and queries to extract rules from trained neural networks,” in: Cohen, W. and Hirsh, H. (eds.), *Machine Learning: Proceedings of the Eleventh International Conference on Machine Learning ML-94* (Rutgers University, NJ), San Francisco, CA: Morgan Kaufmann, pp. 37-45.

De Garis, H. (2002), “What Happened to the ‘CAM-Brain Machines’ (CBMs)?” At http://www.cs.usu.edu/ degaris/news/CBMdown.html (accessed Nov. 2002).

Dinerstein, J., Dinerstein, N., and De Garis, H. (2003), “Automatic multi-module neural network evolution in an artificial brain,” in: *NASA/DoD Conference on Evolvable Hardware (EH’03)* (Chicago, Illinois, USA, 9-11 July), pp. 283-286.

Elizondo, D. and Fiesler, E. (1997), “A survey of partially connected neural networks,” *The International Journal of Neural Systems*, vol. 8, no. 5/6, pp. 535-558.

Engelbrecht, A.P. and Cloete, I. (1996), “A sensitivity analysis algorithm for pruning feedforward neural networks,” in: *IEEE International Conference on Neural Networks* (Washington), vol. 2, pp. 1274-1277.

ERA Technology Ltd. (1997), “Neural networks producing dependable systems,” ERA report 97-0365, Leatherhead. Available at http://www.era.co.uk/techserv /pubs/p970365.htm (accessed 2 May 2001).

Feng, T.-J., Houkes, Z., Spreeuwers, L.J., and Korsten, M.J. (1992), “Internal measuring models in trained networks for parameter estimation from images,” in: *Proceedings of the IEE 4th International Conference on Image Processing and its Applications* (Maastricht, The Netherlands), pp. 230-233.

Footman, T. (ed.) and Young, M.C. (2001), *Guinness Book of World Records*, Bantam Books, p. 126: “Most complex brain building machine.”

Fritsch, T., Mittler, M., and Tran-Gia, P. (1993), “Artificial neural net applications in telcommunication systems,” *Neural Computing & Applications*, vol. 1.

Fu, L. and Chen, T. (1993), “Sensitivity analysis for input vector in multilayer feedforward neural networks,” in: *Proceedings IEEE International Conference on Neural Networks* (San Francisco, CA), part I, pp. 215-218.

Georgakis, A., Kotropoulos, C., Xafopoulos, A., and Pitas, I. (2001), "MM-WEBSOM: a variant of WEBSOM based on order statistics," in: *Proceedings IEEE-EURASIP Workshop Nonlinear Signal and Image Processing* (Baltimore, U.S.A., June).

Giles, C.L., Lawrence, S., and Tsoi, A.C. (2001), "Noisy time series prediction using a recurrent neural network and grammatical inference," *Machine Learning*, vol. 44, no. 1/2 (July/Aug.), pp. 161-183.

Golea, M. (1996), "On the complexity of rule extraction from neural networks and network querying," in: Andrews, R. and Diederich, J. (eds.), *Rules and Networks – Proceedings of the Rule Extraction from Trained Artificial Neural Networks Workshop, Society for the Study of Artificial Intelligence and the Simulation of Behavior Workshop Series (AISB'96)* (University of Sussex, Brighton, U.K., 2nd April), pp. 51-59.

Griffith, N. and Todd, P.M. (eds.) (1999), *Musical Networks: Parallel Distributed Perception and Performance*, Cambridge, MA: The MIT Press.

Grossberg, S. (1973), "Contour enhancement, short term memory, and constancies in reverberating neural networks," *Studies in Applied Mathematics*, vol. 52, pp. 213-257. Reprinted in Grossberg S. (1982), *Studies of Mind and Brain*, Boston, MA: Reidel.

Hamker, F.H. (2001), "Life-long learning cell structures – continuously learning without catastrophic interference," *Neural Networks*, Elsevier Science, vol. 14, no. 4-5, pp. 551-573.

Hammerstrom, D. (2000), "Computational Neurobiology Meets Semiconductor Engineering," Invited Paper, *Multi-Valued Logic Conference 2000*, (Portland, OR, May).

Han, K. and Veloso, M. (1997), "Physical model based multi-objects tracking and prediction in RoboSoccer," in: *Working Note of the AAAI 1997 Fall Symposium*, AAAI, MIT Press.

Hashem, S. (1992), "Sensitivity analysis for feedforward artificial neural networks with differentiable activation functions," in: *Proceedings of the 1992 International Joint Conference on Neural Networks*, Piscataway, NJ: IEEE Press, vol. 1, pp. 419-424.

Haykin, S. (1994), *Neural Networks: a Comprehensive Foundation*, New York: MacMillan.

Hinton, G.E., Sejnowski, T.J., and Ackley, D.H. (1984), “Boltzmann Machines: Constraint Satisfaction Networks That Learn,” Technical report CMU-CS-84-119, Carnegie-Mellon University, Pittsburgh, PA.

Hopfield, J.J. (1982), “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings National Academy of Science*, vol. 79, pp. 2554-2558.

Iordanova, I., Rialle, V., and Vila, A. (1992), “Use of unsupervised neural networks for classification tasks in electromyography,” in: *14th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, vol. 14, pp. 1014-1015.

Jagielska, I. and Matthews, C. (1995), “Fuzzy rule extraction from a trained multilayered neural network,” in: *Proceedings of the IEEE International Conference on Neural Networks* (Perth, Australia, 27 Nov.-1 Dec.), pp. 744-748.

Jain, L.C. and Kacprzyk, J. (2002), *New Learning Paradigms in Soft Computing*, Springer Verlag.

Jain, R., Abraham, A., Faucher, C., and Van der Zwaag, B.J. (eds.) (2003), *Innovations in Knowledge Engineering*, Adelaide, South Australia: Advanced Knowledge International, ISBN 0-9751004-0-8.

Kalman, R.E. and Bucy, R.S. (1961), “New results in linear filter and prediction theory,” *Journal of Basic Engineering*, March issue, pp. 95-108.

Keegstra, H. et al. (1996), “Exploiting network redundancy for lowest-cost neural network realizations,” *Digest ICNN’96* (Washington D.C.), pp. 951-955.

Kim, K. and Bartlett, E.B. (1996), “Nuclear power plant fault diagnosis using neural networks with error estimation by series association,” *IEEE Transactions on Nuclear Science*, vol. 43, no. 4 (August), pp. 2372-2388.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983), “Optimization by simulated annealing,” *Science*, vol. 220, pp. 671-680.

Klimasauskas, C.C. (1991), “Neural nets tell why,” *Dr. Dobbs’s Journal*, pp. 16-24.

Kohle, M. and Schonbauer, F. (1989), “Experience gained with a neural network that learns to play bridge,” in: *Proceedings of the 5th Austrian Artificial Intelligence Meeting*, pp. 224-229.

Kohonen, T. (1982), “Self-organized formation of topologically correct feature maps,” *Biol. Cybern.*, vol. 43, pp. 59-69.

Kohonen, T. (1986), “Learning vector quantization for pattern recognition,” Report TKK-F-A601, Helsinki University of Technology, Espoo, Finland.

Kohonen, T. (1988), “Learning vector quantization,” *Neural Networks*, vol. 1 (suppl 1), pp. 303.

Kohonen, T. (1990), “Improved versions of learning vector quantization,” in: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)* (San Diego, California), vol. 1, pp. 545-550.

Kohonen, T. (1995), *Self-Organizing Maps*, Berlin: Springer-Verlag.

Kosko, B. (1988), “Bidirectional associative memories,” *IEEE Transactions on Systems, Man, Cybernetics*, vol. SMC-18, pp. 49-60.

Kosko, B. (1992), *Neural Networks and Fuzzy Systems*, Englewood Cliffs, NJ: Prentice-Hall.

Kurd, Z. and Kelly, T. (2003), “Establishing safety criteria for artificial neural networks,” in: Palade, V., Howlett, R.J., and Jain, L.C. (eds.), *Proceedings of KES2003* (Oxford, U.K., 3-5 Sep.), part I, *Springer LNAI* 2773, pp. 163-169.

Lam, H.K., Leung, F.H.F., and Tam, P.K.S. (2002), “A switching controller for uncertain non-linear systems,” *IEEE Contr. Syst. Mag.*, vol. 22, no. 1, pp. 7-14.

LeCun, Y., Simard, P.Y., and Pearlmetter, B. (1993), “Automatic learning rate maximization by on-line estimation of the Hessian’s eigenvectors,” in: Hanson, S.J., Cowan, J.D., and Giles, C.L. (eds.), *Advances in Neural Information Processing Systems 5*, San Mateo, CA: Morgan Kaufmann, pp. 156-163.

Lozowski, A., Cholewo, T.J., and Zurada, J.M. (1996), “Symbolic rule representation in neural network models,” in: *Proceedings of the Second Conference on Neural Networks and Their Applications* (Szczyrk, Poland, April), volume 2, pp. 300-305.

McCulloch, W. and Pitts, W. (1943), “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133.

Medsker, L.R. and Jain, L.C. (eds.) (2000), *Recurrent Neural Networks: Design and Applications*, Boca Raton, FL: CRC Press, ISBN 0-8493-7181-3.

Meijer, P.B.L. (1996), *Neural Network Applications in Device and Sub-circuit Modeling for Circuit Simulation*, Ph.D. thesis, Eindhoven University, the Netherlands.

Miller, W.T. (1994), “Real-time neural network control of a biped walking robot,” *IEEE Control Systems*, vol. 14, Feb. issue, pp. 41-48.

Minsky, M. and Papert, S.A. (1969), *Perceptrons*, MIT-Press, Cambridge MA.

Mitchell, J. (1992), “A geometric interpretation of hidden layer units in feedforward neural networks,” *Network*, vol. 3, pp. 19-25.

Muhamad, A.K. and Deravi, F. (1994), “Neural networks for the classification of image texture,” *EngAAI*, vol. 7, no. 4, pp. 381-393.

Narendra, K.S. and Parthasarathy, K. (1990), “Identification and control of dynamical systems using neural networks,” *IEEE Tr. NN*, vol. 1, no. 1, pp. 4-27.

Navone, H. and Ceccatto, H. (1994), “Predicting Indian monsoon rainfall: a neural network approach,” *Climate Dynamics*, vol. 10, pp. 305-312.

“Classification and evaluation of algorithms for rule extraction from artificial neural networks,” Manuscript (August 1998), Division of Informatics, University of Edinburgh, U.K.

NIWI (Netherlands Institute for Scientific Information Services) (2001), “Project: EUREGIO Computational Intelligence Center for SME,” *Dutch Research Database*. At http://www.niwi.nl/en/oi/nod/onderzoek/OND1268254/toon (16 Nov. 2001).

Oja, M., Kaski, S., and Kohonen, T. (2003), “Bibliography of Self-Organizing Map (SOM) papers: 1998-2001 addendum,” *Neural Computing Surveys*, vol. 3, pp. 1-156.

Olden, J.D. and Jackson, D.A. (2002), “Illuminating the ‘black box’: a randomization approach for understanding variable contributions in artificial neural networks,” *Ecological Modelling*, Elsevier Science, vol. 154, pp. 135-150.

Palade, V., Neagu, D.-C., and Patton, R.J. (2001), “Interpretation of trained neural networks by rule extraction,” in Reusch, B. (ed.), *Computational Intelligence: Theory and Applications, Proceedings of the International Conference 7th Fuzzy Days* (Dortmund, Germany, Oct.), *Springer LNCS* vol. 2206, pp. 152-161.

Principe, J., Giles, L., Morgan, N., and Wilson, E. (eds.) (1997), *IEEE Workshop on Neural Networks for Signal Processing VII*, IEEE Press.

Rios Insua, D. (1990), *Sensitivity Analysis in Multi-Objective Decision Making*, vol. 347 of *Lecture Notes in Economics and Mathematical Systems*, Berlin: Springer Verlag.

Ritter, H., Martinetz, T., and Schulten, K. (1992), *Neural Computation and Self-Organizing Maps*, Reading, MA: Addison-Wesley.

Rosario, B. and Hearst, M. (2001), "Classifying the semantic relations in noun compounds via a domain-specific lexical hierarchy," in: *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP '01),* (Pittsburg, PA, June).

Rosenblatt, F. (1958), "The Perceptron: a probabalistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, pp. 386-408.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986a), "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536.

Rumelhart, D.E., McClelland, J.L., et al. (1986b), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol 1: *Foundations*, Cambridge, MA: MIT Press.

Ruppin, E. and Yeshurun, Y. (1991), "Recall and recognition in an attractor neural network model of memory retrieval," manuscript, Department of Computer Science, Tel Aviv University, Israel.

Sarle, W.S. (2001), "Neural Network FAQ." Available by FTP from ftp.sas.com /pub/neural/FAQ.html (version of 15 Feb. 2001).

Schmid, H. (1994), "Part–of–speech tagging with neural networks," in: *Proceedings of the 15th International Conference on Computational Linguistics (COLING)* (Kyoto, Japan), pp. 172-176.

Schmitt, M., Teodorescu, H.-N., Jain, A., Jain, A., Jain, S., and Jain, L.C., (eds.) (2002), *Computational Intelligence Processing in Medical Diagnosis*, Springer Verlag.

Schmitz, G.P.J., Aldrich, C., and Gouws, F.S. (1999), "ANN-DT: an algorithm for extraction of decision trees from artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 10, pp. 1392-1401.

Scott, G.M., Shavlik, J.W., and Ray, W.H. (1992), "Refining PID controllers using neural networks," in: Moody, J., Hanson, S., and Lippmann, R. (eds.), *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann, pp. 555-562.

Setiono, R. and Liu, H. (1995), "Understanding neural networks via rule extraction," in: *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (Montreal, Canada), pp. 480-485.

Sharkey, A.J.C., Sharkey, N.E., and Gopinath, O.C. (1995), "Diversity, neural nets and safety critical applications," in: Niklasson, L.F. and Boden, M.B. (eds.), *Current Trends in Connectionism*, Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 165-178.

Shepherd, G.M. and Koch, C. (1990), “Introduction to synaptic circuits,” in: Shepherd, G.M. (ed.), *The Synaptic Organization of the Brain*, 3rd ed., New York: Oxford University Press, pp. 3-31.

Siegelmann, H.T. (1998), *Neural Networks and Analog Computation: Beyond the Turing Limit*, Boston: Birkhauser.

Sjoberg, J., Zhang, Q., Ljung, L., Benveniste, A., Deylon, B., Glorennec, P.-Y. Hjalmarsson, H., and Juditsky, A. (1995), “Nonlinear black-box modeling in system identification: a unified overview,” *Automatica*, vol. 31, no. 12, pp. 1691-1724.

Spaanenburg, L. (2000), “Knowledge fusion in modular neural networks,” *Proceedings of the NC2000*, pp. 356-362.

Spaanenburg, L. (2001), “Unlearning in feed-forward multi-nets,” *Proceedings ICANNGA’01* (Prague), pp. 106-109.

Spaanenburg, L., Jansen, W.J., and Nijhuis, J.A.G. (1997a), “Over multiple rule-blocks to modular nets,” *Proceedings Euromicro’97*, pp. 698-705.

Spaanenburg, L., Jansen, W.J., and Nijhuis, J.A.G. (1997b), “Injecting functions in neural nets by controlled dedication,” *Proceedings ECCTD’97* (Budapest, Hungary), pp. 1108-1113.

Spaanenburg, L., Ter Haseborg, H.M.G., and Peng, W. (2001), “Trimming neural networks for embedded intelligence,” *International Journal of Knowledge-Based Intelligent Engineering Systems*, vol. 5, no. 3, pp. 171-178.

Specht, D.F. (1988), “Probabilistic neural networks for classification, mapping or associative memory,” in: *ICNN-88 Conference Proceedings*.

Steinbuch, K. and Piske, U.A.W. (1963), “Learning matrices and their applications,” *IEEE Transactions on Electronic Computing*, vol. 12, Dec. issue, pp. 846-862.

Stephanopoulos, G. (1984), *Chemical Process Control: an Introduction to Theory and Practice*, Englewood Cliffs, NJ: Prentice Hall, Inc.

Takefuji, Y. and Szu, H. (1989), “Design of parallel distributed Cauchy machines,” in: *Proceedings of the First International Joint Conference on Neural Networks (IJCNN)* (Washington, June 1989).

Tan, Y., Dang, X., and Van Cauwenberghe, A. (1999), “Generalised nonlinear PID controller based on neural networks,” in: *Proceedings, Information, Decision and Control* (Adelaide, Australia, 8-10 February), IEEE, pp. 519-524.

Tao, Y., Heinemann, P.H., Varghese, Z., Morrow, C.T., and Sommer III, H.J. (1995), "Machine vision for color inspection of potatoes and apples," *Transactions of the American Society of Agricultural Engineers*, vol. 38, no. 5, pp. 1555-1561.

TerBrugge, M.H., Nijhuis, J.A.G., Jansen, W.J., Drenth, H., and Spaanenburg, L. (1995), "On the representation of data for optimal learning," in: *Proceedings ICNN'95* (Perth, Western Australia), part VI, pp. 3180-3184.

Thrun, S.B. (1991), "The Monk's Problems – a Performance Comparison of Different Learning Algorithms," Technical report, Carnegie Mellon University, Pittsburg, PA.

Thrun, S.B. (1993), "Extracting Provably Correct Rules from Artificial Neural Networks," Technical report IAI-TR-93-5, Institut für Informatik III, Universität Bonn, Germany.

Towell, G. (1991), *Symbolic Knowledge and Neural Networks: Insertion, Refinement and Extraction*, Ph.D. thesis, Computer Science Department, University of Wisconsin, Madison.

Trentelman, H.L. and Willems, J.C. (1993), *Essays on Control: Perspectives in the Theory and Its Applications*, Boston: Birkhaeuser.

Trippi, R.R. and Turban, E. (1993), *Neural Networks in Finance and Investing*, Chicago: Probus.

Van der Heijden, F. (1994), *Image Based Measurement Systems*, Chichester, England: John Wiley & Sons.

Van der Steen, E.W., Nijhuis, J.A.G., Spaanenburg, L., and Van Veelen, M. (2001), "Sampling casts plasticity in adaptive batch control," in: *Proceedings ProRISC'01* (Veldhoven, the Netherlands, Nov.), pp. 646-651.

Van der Zwaag, B.J. (2001), "Handwritten digit recognition: a neural network demo," in Reusch, B. (ed.), *Computational Intelligence: Theory and Applications, Proceedings of the International Conference 7th Fuzzy Days* (Dortmund, Germany, Oct.), *Springer LNCS* vol. 2206, pp. 762-771.

Van der Zwaag, B.J. and Slump, C. (2002), "Analysis of neural networks for edge detection," in: *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing* (Veldhoven, the Netherlands, 28-29 Nov.), pp. 580-586.

Van der Zwaag, B.J., Spaanenburg, L., and C. Slump, C. (2002a), "Analysis of neural networks in terms of domain functions," in: *Proceedings IEEE Benelux Signal Processing Symposium SPS-2002* (Leuven, Belgium, 21-22 March), pp. 237-240.

Van der Zwaag, B.J., Slump, C.H., and Spaanenburg, L. (2002b), “Analysis of neural networks through base functions,” in: *Book of Abstracts, Lerende Oplossingen (Learning Solutions)* (Nijmegen, 14 June), STW/SNN, pp. 34-35.

Van der Zwaag, B.J., Slump, K., and Spaanenburg, L. (2002c), “Process identification through modular neural networks and rule extraction,” in: Ruan, D., D’hondt, P., and Kerre, E.E. (eds.), *Computational Intelligent Systems for Applied Research, Proceedings of the 5th International FLINS Conference* (Ghent, Belgium, 16-18 Sept.), Singapore: World Scientific, pp. 268-277.

Van der Zwaag, B.J., Slump, C.H., and Spaanenburg, L. (2002d), “Process identification through modular neural networks and rule extraction,” in: H. Blockeel and M. Denecker (eds.), *Proceedings of the 14th BNAIC* (Leuven, Belgium, 21-22 Oct.), pp. 507-508.

Van der Zwaag, B.J., Slump, K., and Spaanenburg, L. (2003a), “Extracting knowledge from supervised neural networks in image processing,” Chapter 5 in (Jain et al. 2003), pp. 107-127.

Van der Zwaag, B.J., Slump, K., and Spaanenburg, L. (2003b), “On the analysis of neural networks for image processing,” in: Palade, V., Howlett, R.J., and Jain, L.C. (eds.), *Proceedings of KES2003* (Oxford, U.K., 3-5 Sep.), part II, *Springer LNAI* 2774, pp. 950-957.

Van der Zwaag, B.J., Spaanenburg, L., and Slump, C. (2003d), “Translating feedforward neural nets into SOM-like maps,” in: *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing* (Veldhoven, the Netherlands, 26-27 Nov.).

vanVeelen, M., Nijhuis, J.A.G., and Spaanenburg, L. (2000), “Emergence of learning methodology for abnormality detection,” *Proceedings BNAIC’00* (Kaatsheuvel, the Netherlands), pp. 283-292.

Venema, R.S. (1999), *Aspects of an Integrated Neural Prediction System*, Ph.D. thesis, Rijksuniversiteit Groningen, the Netherlands.

Venema, R.S. and Spaanenburg, L. (2001), “Learning feed-forward multi-nets,” *Proceedings ICANNGA’01* (Prague), pp. 102-105.

Verma, B., Blumenstein, M., and S. Kulkarni, S. (1997), “A neural network based technique for data compression,” in: *Proceedings of the IASTED International Conference on Modelling and Simulation (MOS ’97)*, Singapore: IASTED Press, pp. 12-16.

Vonk, E., Jain, L.C., Veelenturf, L.P.J., and Johnson, R. (2000), "Automatic generation of a neural network architecture using evolutionary computation," *Proceedings of Electronic Technology Directions to the Year 2000* (Adelaide, South Australia, 23-25 May), Los Alamitos, CA: IEEE Computer Society Press, pp. 142-147.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K.J. (1989), "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustic, Speech and Signal Processing*, ASSP-37, pp. 328-339.

Wejchert, J. and Tesauro, G. (1991), "Visualizing processes in neural networks," *IBM Journal of Research and Development*, vol. 35, pp. 244-253.

Wentink, M. (1995), "Master-Slave Kohonen Network Applied to Digit Recognition," Internal Report BSC-021N95, University of Twente, Enschede, the Netherlands.

Werbos, P. (1974), *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. thesis, Harvard University, Comittee on Applied Mathematics.

Wermter, S. (2000), "Knowledge extraction from transducer neural networks," *Journal of Applied Intelligence*, Boston: Kluwer Academics, vol. 12, pp. 27-44.

Widrow, B. and Hoff, M.E. (1960), "Adaptive switching circuits," *WESTCON Convention Record, Part IV*, pp. 96-104.

Williams, R.J. and Zipser, D. (1989), "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 270-280.

Worth, A.J. and Spencer R.R. (1989), "A neural network for tactile sensing: the Hertzian contact problem," in: *International Joint Conference on Neural Networks*, part I, pp. 267-274.

Zhang, W., Giger, M., Nishikawa, R., and Doi, K. (1994), "Application of a shift-invariant artificial neural network for detection of breast carcinoma in digital mammograms," in: *World Congress on Neural Networks (WCNN'94)* (San Diego, CA), vol. I, pp. 47-52.

Zurada, J.M. (1992), *Introduction to Artificial Neural Systems*, Boston: PWS.

# Summary

Since the early development of artificial neural networks, researchers have tried to analyze trained neural networks in order to gain insight into their behavior. For certain applications and in certain problem domains this has been successful, for example by the development of so-called rule extraction and other methods. For many other problem domains the existing knowledge extraction methods do not offer satisfactory solutions. Another key factor involved in the knowledge extractability is formed by the neural network architecture, as some network architectures (e.g., self-organizing maps) offer more direct insight into the stored knowledge than others (e.g., feedforward nets).

This thesis therefor presents a generic neural network analysis method that utilizes domain-specific basic functions that are easy to interpret by the user and that can furthermore be used to optimize neural network systems.

In general, the analysis consists in describing the internal functionality of the neural network in terms of domain-specific basic functions, functions that can be considered basic in the application domain of the neural network. This means that users who may not be familiar with artificial neural networks, but who are familiar with basic functions that are often used in their problem domain, can gain insight in the way the neural network solves their problem. For such users, this is often an important factor in deciding to apply artificial neural networks to a problem that may be difficult to solve otherwise.

Traditionally, artificial neural network systems are monolithic, single-tier systems. In such systems, the process knowledge is distributed among all elements of the system. This makes it very difficult to extract the contained knowledge. It is much easier to locate knowledge in a system if it is based on a 3-tier model. Basic domain knowledge is stored in the elementary functions that are found in the bottom tier. It can be easily identified or analyzed. The middle tier basically describes how this knowledge is used in the system, and the application results, that are acquired on the base of the system knowledge, are presented in the top tier.

Therefor, it would be beneficial in many aspects if systems based on a single-tier model could be transformed into equivalent systems based on the 3-tier model. This thesis proposes a generic method that can be used as a first step to achieve this for monolithic neural network systems. Whereas in general the system knowledge is stored in the neural network in a distributive manner, this thesis shows that it is possible to create a foundation tier on which the whole system is apparently based.

The method presented in this thesis breaks the internal system knowledge into identifiable basic foundation blocks. These foundation blocks, or basic functions, depend on the application domain in which the system is operational, and so do the methods to extract those basic functions. However, the domain-dependent methods are all based on a single generic domain-independent idea, namely the analysis of the neural network in terms of (generic) domain-dependent basic functions.

This thesis first gives a brief overview of the enormous variety of neural networks and applications that have been developed since the first mathematical model of human nerve cells was described. This is followed by a review of existing methods for the analysis of neural networks. This includes a discussion to what extent these existing methods are successful in retrieving complete and comprehensive knowledge from the network.

A study is presented which identifies those application domains for which existing knowledge extraction techniques produce insufficient results. This is then used as a starting point for exploring the suitability of existing and new methods for neural network analysis in those domains. This is then further expanded by outlining the new theory of the analysis of trained neural networks in terms of domain-dependent basic functions, the main topic in this thesis. Based on this, suggestions for basic functions are given for a range of application domains.

Two of these suggestions are then worked out in more detail and applied to some typical applications in the respective problem areas. One example application that is worked out is edge detection, from the digital image processing domain. The stored domain knowledge is translated into sets of gradient filters, which are commonly used in image processing. Another one is a feedforward network for classification of character images. Class knowledge stored in this network is extracted in the form of class prototypes. It is expected that these examples offer good illustrations of the benefits of the new method for neural network analysis presented in this thesis.

The method presented in this thesis offers a much wider applicability than existing methods, some of which can actually be defined as specific cases of the generic method that forms the central theme in this thesis.

# Samenvatting

Sinds de eerste ontwikkeling van kunstmatige neurale netwerken hebben onderzoekers geprobeerd getrainde neurale netwerken te analyseren om zo inzicht te krijgen in hun gedrag. Voor bepaalde toepassingen en in bepaalde toepassingsgebieden is dit geslaagd, bijvoorbeeld door de ontwikkeling van zogenoemde regelextractie en andere methoden. Voor veel andere toepassingsgebieden bieden de bestaande methoden geen bevredigende oplossingen. Een andere belangrijke factor wordt gevormd door de architectuur van het neurale netwerk, omdat bepaalde netwerk architecturen (bijv. *self-organizing maps*) meer rechtstreeks inzicht bieden in de opgeslagen kennis dan sommige andere (bijv. *feedforward* netwerken).

Daarom presenteert dit proefschrift een generieke neurale netwerkanalysemethode die gebruik maakt van domein-specifieke basisfuncties die gemakkelijk te interpreteren zijn door de gebruiker en die bovendien gebruikt kunnen worden voor de optimalisatie van neurale netwerksystemen. Globaal gezien bestaat de analyse uit een beschrijving van de interne functionaliteit van het neurale netwerk in termen van functies die kunnen worden beschouwd als basisfuncties binnen het toepassingsdomein van het neurale netwerk. Dit betekent dat gebruikers die misschien niet bekend zijn met kunstmatige neurale netwerken, maar die wel bekend zijn met basisfuncties die veel gebruikt worden binnen hun toepassingsgebied, inzicht kunnen krijgen in de manier waarop het neurale netwerk hun probleem oplost. Voor zulke gebruikers is dit vaak een belangrijke factor bij het besluiten om kunstmatige neurale netwerken toe te passen op een anderszins moeilijk oplosbaar probleem.

Kunstmatige neurale netwerken zijn traditioneel monolithische, enkel-niveau systemen. In zulke systemen wordt de proceskennis gedistrubueerd over alle elementen van het systeem. Dit maakt het erg moeilijk om de bevatte kennis te extraheren. Het is veel gemakkelijker om kennis in het syteem te lokaliseren als het op een 3-niveaus model is gebaseerd. Fundamentele domeinkennis wordt opgeslagen in de elementaire functies die zich bevinden op het onderste niveau. Deze kennis kan gemakkelijk worden geanalyseerd. Het middelste niveau beschrijft hoe deze kennis in het systeem wordt gebruikt, en het bovenste niveau presenteert de toepassingsresultaten die op basis hiervan worden verkregen.

Het zou dus goed zijn als systemen die zijn gebaseerd op een enkel-niveau model zouden kunnen worden omgezet in gelijkwaardige systemen gebaseerd op het 3-niveaus model. Dit proefschrift stelt een generieke methode voor die gebruikt kan worden als een eerste stap om dit te bereiken voor monolithische neurale netwerk-systemen. Terwijl in het algemeen de systeemkennis op een distributieve wijze wordt opgeslagen in het neurale netwerk, laat dit proefschrift zien dat het mogelijk is een fundamenteel niveau te creëren waarop het hele systeem kennelijk is gebaseerd.

Deze nieuwe methode breekt de interne systeemkennis in identificeerbare basisblokken. Deze fundamentele blokken, of basisfuncties, hangen af van het toepassingsdomein, en hetzelfde geldt voor de methoden om deze basisfuncties te extraheren. Deze domeinafhankelijke methoden zijn echter alle gebaseerd op een enkel generiek domeinonafhankelijk idee, zijnde de analyse van het neurale netwerk in termen van (generieke) domeinafhankelijke basisfuncties.

Dit proefschrift geeft een kort overzicht van de enorme variatie aan neurale netwerken en hun toepassingen. Dit wordt gevolgd door een bespreking van bestaande methoden voor neurale netwerkanalyse, inclusief in hoeverre deze bestaande methoden succesvol zijn in het achterhalen van geheelomvattende kennis uit het netwerk.

Hierop volgt een studie ter identificatie van die toepassingsgebieden waarvoor bestaande kennisextractietechnieken onbevredigende resultaten geven. Dit wordt gebruikt als een startpunt om te onderzoeken welke bestaande en nieuwe methoden voor neurale netwerkanalyse geschikt kunnen zijn voor die gebieden. Dit wordt dan verder uitgebreid voor een uiteenzetting over de nieuwe theorie van de analyse van getrainde neurale netwerken in termen van domeinspecifieke basisfuncties, het hoofdonderwerp van dit proefschrift. Op basis hiervan worden suggesties gegeven voor een scala van toepassingsdomeinen.

Een uitgewerkte voorbeeldtoepassing is *edge*-detectie, uit het digitale beeldbewerkingsdomein. De opgeslagen domeinkennis wordt vertaald naar een verzameling gradiëntfilters, die algemeen worden gebruikt in beeldbewerking. Een tweede voorbeeld is een *feedforward* netwerk voor classificatie van letters. Opgeslagen klassekennis wordt uit het netwerk afgeleid in de vorm van klasseprototypes. De verwachting is dat deze voorbeelden een goed beeld geven van de voordelen van de in dit proefschrift gepresenteerde nieuwe neurale netwerkanalysemethode.

De in dit proefschrift gepresenteerde kennisextractiemethode biedt veel bredere toepassingsmogelijkheden dan bestaande methoden, waarvan overigens een aantal kan worden gedefinieerd als bijzondere gevallen van de generieke methode die in dit proefschrift centraal staat.

# Biography

Berend Jan van der Zwaag received his M.Sc. degree in Electrical Engineering from the University of Twente in 1995. After that he worked in the School of Electrical Engineering and in the School of Computer and Information Science at the University of South Australia for several years. First in the Knowledge-Based Intelligent Engineering Systems Centre (KES), and later in the Advanced Computing Research Centre (ACRC). During the past four years he has been employed by the Department of Electrical Engineering at the University of Twente. Initially in the EUREGIO Computational Intelligence Centre (ECIC), and later in the Centre for Telematics and Information Technology (CTIT). He has co-authored over 20 papers in international journals and conferences, and has provided editorial assistance for over 30 books in the field of computational intelligence. His research interests are mainly in computational intelligence, particularly neural networks, fuzzy logic, genetic algorithms, intelligent user interfaces, and biometric systems, as well as in image processing and medical signal processing. Apart from this he has a side interest in cellular automata and in recreational mathematics, a result of which can be seen on the cover of this work.

# List of publications

1. A. Abraham, L. Jain, and B.J. van der Zwaag (eds.) (2004), *Innovations in Intelligent Systems: Design, Management and Applications*, vol. 140 of *Studies in Fuzziness and Soft Computing*, Heidelberg: Springer Verlag, ISBN 3-540-20265-X. To appear in January.

2. S. Malki, L. Spaanenburg, and B.J. van der Zwaag (2003), “It takes a winner to take his share,” in: *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing* (Veldhoven, the Netherlands, 26-27 Nov.).

3. M.-J. Hoeve, B.J. van der Zwaag, M. van Burik, K. Slump, and R. Jones (2003), “Detecting epileptic seizure activity in the EEG by independent component analysis,” in: *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing* (Veldhoven, the Netherlands, 26-27 Nov.).

4. B.J. van der Zwaag, L. Spaanenburg, and K. Slump (2003), “Translating feedforward neural nets to SOM-like maps,” in: *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing* (Veldhoven, the Netherlands, 26-27 Nov.).

5. B.J. van der Zwaag, S. Malki, and L. Spaanenburg (2003), “Maturing a network structure for rule extraction,” in: *Book of Abstracts, Lerende Oplossingen (Learning Solutions)* (Nijmegen, 22 Oct.), STW/SNN.

6. B.J. van der Zwaag, K. Slump, and L. Spaanenburg (2003), “On the analysis of neural networks for image processing,” in: Palade, V., Howlett, R.J., and Jain, L.C. (eds.), *Proceedings of KES2003* (Oxford, U.K., 3-5 Sep.), part II, *Springer LNAI* 2774, pp. 950-957.

7. R. Jain, A. Abraham, C. Faucher, and B.J. van der Zwaag (eds.) (2003), *Innovations in Knowledge Engineering*, Adelaide, South Australia: Advanced Knowledge International, ISBN 0-9751004-0-8.

8. B.J. van der Zwaag, K. Slump, and L. Spaanenburg (2003), “Extracting knowledge from supervised neural networks in image processing,” Chapter 5 in (Jain et al. 2003), pp. 107-127.

9. L. Spaanenburg, R. Alberts, C.H. Slump, and B.J. vanderZwaag (2003), “Natural learning of neural networks by reconfiguration,” in: A. Rodriguez-Vazquez, D. Abbott, and R. Carmona (eds.), *Bioengineered and Bioinspired Systems* (Gran Canaria, Spain, 19-21 May), *Proceedings of SPIE* vol. 5119, pp. 273-284.

10. B.J. van der Zwaag and C. Slump (2002), “Analysis of neural networks for edge detection,” in: *Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing* (Veldhoven, the Netherlands, 28-29 Nov.), pp. 580-586.

11. B.J. van der Zwaag, C.H. Slump, and L. Spaanenburg (2002), “Process identification through modular neural networks and rule extraction,” in: H. Blockeel and M. Denecker (eds.), *Proceedings of the 14th BNAIC* (Leuven, Belgium, 21-22 Oct.), pp. 507-508.

12. B.J. van der Zwaag, K. Slump, and L. Spaanenburg (2002), “Process identification through modular neural networks and rule extraction,” in: D. Ruan, P. D’hondt, and E.E. Kerre (eds.), *Computational Intelligent Systems for Applied Research, Proceedings of the 5th International FLINS Conference* (Ghent, Belgium, 16-18 Sept.), Singapore: World Scientific, pp. 268-277.

13. B.J. van der Zwaag, C.H. Slump, and L. Spaanenburg (2002), “Analysis of neural networks through base functions,” in: *Book of Abstracts, Lerende Oplossingen (Learning Solutions)* (Nijmegen, 14 June), STW/SNN, pp. 34-35.

14. L. Spaanenburg, S. Achterop, C.H. Slump, and B.J. van der Zwaag (2002), “Molding the knowledge in modular neural networks,” in: *Book of Abstracts, Lerende Oplossingen (Learning Solutions)* (Nijmegen, 14 June), STW/SNN, pp. 25-26.

15. B.J. van der Zwaag, L. Spaanenburg, and C. Slump (2002), “Analysis of neural networks in terms of domain functions,” in: *Proceedings IEEE Benelux Signal Processing Symposium SPS-2002* (Leuven, Belgium, 21-22 March), pp. 237-240.

16. L. Spaanenburg, C. Slump, R. Venema, and B.J. van der Zwaag (2002), “Preparing for knowledge extraction in modular neural networks,” in: *Proceedings IEEE Benelux Signal Processing Symposium SPS-2002* (Leuven, Belgium, 21-22 March), pp. 121-124.

17. E. Friedman and B.J. van der Zwaag (2002), "Constant neighbor dihedral tilings with 15, 32, and 43 neighbors," *Geombinatorics*, vol. XI, no. 3 (Jan.), pp. 74-77.

18. B.J. van der Zwaag (2001), "Handwritten digit recognition: a neural network demo," in B. Reusch (ed.), *Computational Intelligence: Theory and Applications, Proceedings of the International Conference 7th Fuzzy Days* (Dortmund, Germany, Oct.), *Springer LNCS* vol. 2206, pp. 762-771.

19. J.R. Warren, J.T. Noone, B.J. Smith, R. Ruffin, P. Frith, B.J. van der Zwaag, G.V. Beliakov, H.K. Frankel, and H.J. McElroy (2001), "Automated attention flags in chronic disease care planning," *Medical Journal of Australia*, vol. 175, 17 Sept., pp. 308-312.

20. A.M. Bazen, G.T.B. Verwaaijen, S.H. Gerez, L.P.J. Veelenturf, and B.J. van der Zwaag (2000), "A correlation-based fingerprint verification system," in: *Proceedings of ProRISC 2000* (Veldhoven, the Netherlands, 30 Nov. - 1 Dec.), pp. 205-213.

21. J. Warren, G. Beliakov, and B. van der Zwaag (2000), "Fuzzy logic in clinical practice decision support systems," in: *Proceedings of the 33rd Hawaii International Conference on Systems Sciences*, IEEE Computer Society Press, January 2000.

22. J.R. Warren, H.K. Frankel, J.T. Noone, and B.J. van der Zwaag (2000), "Supporting special-purpose health care models via Web interfaces," in: *Proceedings of the First Australasian User Interface Conference*, IEEE Computer Society Press, Jan. 2000, pp. 118-125.

23. B.J. van der Zwaag and D. Corbett (1999), "Capturing hand tremors with a fuzzy logic wheelchair joystick controller," *Biomedical Fuzzy and Human Sciences*, Official Journal of the Biomedical Fuzzy Systems Association, vol. 5, no. 1, September, pp. 49-55.

24. B.J. van der Zwaag, D. Corbett, and L.C. Jain (1999), "Minimizing tremor in a joystick using fuzzy logic," in: *Proceedings of the Third International Conference on Knowledge-Based Intelligent Information Engineering Systems (KES'99)*, IEEE Press, USA, pp. 5-8.

25. D. Corbett and B.J. van der Zwaag (1998), "Experiments with the fuzzy logic controlled wheelchair," in: *Proceedings of the 5th International Conference on Soft Computing and Information/Intelligent Systems (IIZUKA'98)*, pp. 459-462.

26. D.R. Corbett and B.J. van der Zwaag (1998), “Using fuzzy logic to mitigate the effect of multiple-sclerosis tremors on a wheelchair joystick controller,” in: *Proceedings of the 11th Australian Joint Conference on Artificial Intelligence* (Brisbane, Australia), *Springer Lecture Notes in Computer Science*, vol. 1502, Springer-Verlag, pp. 39-46.

27. L.J. Spreeuwers, B.J. van der Zwaag, and F. van der Heijden (1995), “Context dependent learning in neural networks,” in: *Proceedings of the Fifth International Conference on Image Processing and Its Applications* (Edinburgh, UK, July 4-6), pp. 632-636.