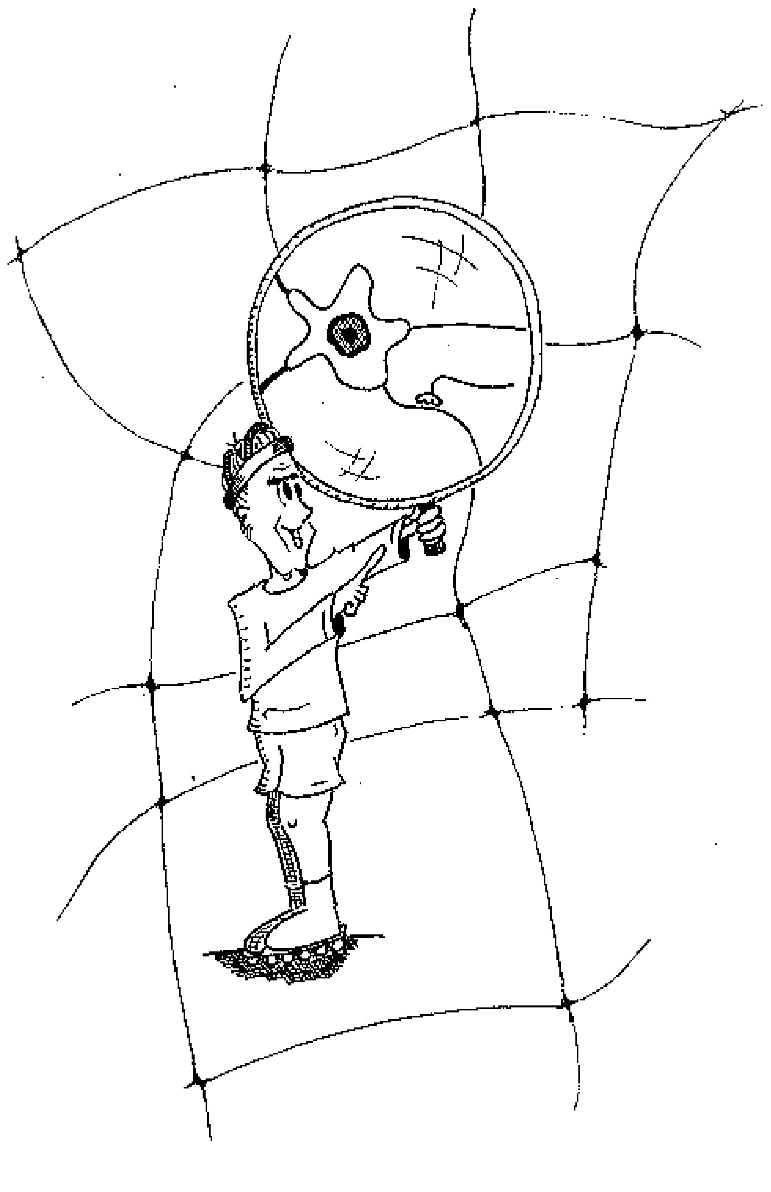
Generalisation in Feed Forward Neural Classifiers



Aarnoud Hoekstra

Generalisation in Feed Forward Neural Classifiers

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.F. Wakker,
in het openbaar te verdedigen ten overstaan van een commissie,
door het College voor Promoties aangewezen,
op dinsdag 20 oktober 1998 te 16.00 uur

door

Aarnoud HOEKSTRA

ingenieur informatica
geboren te Sneek

Dit proefschrift is goedgekeurd door de promotor:
prof. dr. I.T. Young.

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. I.T. Young,	Technische Universiteit Delft, promotor
Dr. ir. R.P.W. Duin,	Technische Universiteit Delft, toegevoegd promotor
Prof. dr. ir. E. Backer,	Technische Universiteit Delft
Prof. dr. E.S. Gelsema,	Erasmus Universiteit Rotterdam
Prof. dr. N.J. Kok,	Rijksuniversiteit Leiden
Dr. ir. B.J.A. Kröse,	Universiteit van Amsterdam
Dr. H.J. Kappen,	Katholieke Universiteit Nijmegen
Prof. dr. H. Koppelaar,	Technische Universiteit Delft, reservelid

NWO SION

This research has been sponsored by the dutch foundation for scientific research in the Netherlands (NWO) and the dutch foundation for computer science research (SION).

Advanced School for Computing and Imaging

This work was carried out in graduate school ASCI. ASCI dissertation series number 33.

ISBN : 90-9012050-5

”Wat vandaag algemeen bekend is, was gisteren wetenschap.”

Niels Bohr

Contents

Summary

Generalisation in Feed Forward Neural Classifiers

The use of neural networks, or neural classifiers as they are also referred to, has become common practice in the pattern recognition practice. Neural networks are considered to be very powerful classifiers compared to classical algorithms such as the nearest neighbour method. The algorithms used in neural network applications are capable of finding a good classifier based on a limited and in general a small number of training examples. This capability, also referred to as generalisation, is of interest from a pattern recognition point of view since a large set of parameters is estimated using a relatively small data set. In this thesis the generalisation behaviour of neural networks is studied. In particular, the question of how this behaviour can be detected and which factors influence it are answered.

To be able to answers these questions, a proper understanding of the concept of *generalisation* is needed, such that the results obtained from the introduced techniques can be compared. Therefore, an operational definition of generalisation is introduced, namely the number of expected errors made by a classifier on a set of test samples. The set of neural classifiers studied in this thesis was restricted to the class of backpropagation trained classifiers and classical algorithms as the *k* nearest neighbour method, linear and quadratic classifiers.

The first objective is to gain more insight into the behaviour of a neural network. Consecutively, a measure can be applied which holds for both neural and classical classifiers. By using a nonlinearity measure, insight is obtained in the generalisation behaviour of a neural network. This measure shows to which extent the network has adapted to the data set. The better it adapts to the data, the smaller its generalisation error. Too much adaptation to the data implying a high nonlinearity, however, results in a non-generalising classifier. By monitoring the value of the nonlinearity measure during training this might be avoided. The definition of the nonlinearity measure is such that it also applies to non-neural classifiers. This enables us to compare classical classifiers and neural networks. It shows that neural networks start from a linear solution and gradually adapt in a nonlinear fashion. This adaptation is even stronger, and hence resulting in a larger nonlinearity, for a neural classifier than a classical one. It might therefore indicate that a neural network has a larger effective capacity compared to the classical classifiers.

Another factor which influences the generalisation behaviour, is the architecture of a network. In this thesis the architecture is determined by the number of hidden units. If the number of hidden units is larger than needed, but not too large, the network will learn faster. This is due to the redundancy present in the network. Redundancy causes the neurons to cluster. At the start of a learning cycle, neurons fullfil approximately the same functions. As the training progresses, the neurons start to specialise. This specialisation, known as symmetry breaking, can be visualised using a projection technique which maps the high dimensional space in which the hidden units move onto a two dimensional plane. In this two dimensional plane the trajectories of the hidden units during training can be depicted in order to understand the training behaviour.

Finally, the reliability of a neural network classification was studied. After the training cycle one is interested in the reliability of the classifications made by a neural network. This is done by estimating the a posteriori probabilities of the classifications. These probabilities can be used to improve the network reliability by creating better networks or used in rejecting samples. We can estimate the probabilities by using confidence value estimators. These estimators can be determined using three techniques: the network outputs, the nearest neighbour method and the logistic estimator. Using these estimators, the reliability of a network classification is checked. A problem, however, is that these estimators need an independent test set. Such a set can only be used once in order to prevent obtained biased classifiers. This problem was circumvented by the introduction of *k* nearest neighbour data generation method. Using this method a new set is generated from a learning set. This new set, referred to as validation set, can act as a substitute of a test set.

After applying the techniques presented in this thesis, it can be concluded that we have gained more insight into the generalisation behaviour of neural classifiers. In particular the nonlinearity measure is of interest since it enables the comparison of neural and non-neural classifiers in an objective manner.

Aarnoud Hoekstra, 1998

Chapter 1

Introduction

1.1 Neural networks

Mankind has always been interested in how the human brain actually works. From the ancient Greek philosophers to contemporary researchers, many attempts have been made to make a model of the human brain. However, it is not yet impossible. The human brain is a very powerful instrument for all kinds of tasks, from vision and hearing to steering the muscles. Probably the most interesting feature is its capability to learn and to anticipate new tasks. Therefore, by modelling the brain using some artificial model, for example by a computer simulation, one might obtain this same functionality. In our research we do not aim at understanding the working of the human brain, but instead take advantage of the models which simulate the brain's functionality and apply them in solutions. We are interested in models which are able to learn and even generalise, i.e. able to cope with new environments. These models which are artificial constructs modelled in a computer environment are referred to as artificial neural network models.

In order to understand the development of artificial neural networks as used in current research, one has to have some understanding of how the biological brain works. This is necessary since the concepts of artificial neural nets are derived from biological research. The (human) brain consists of basic elements called *neurons*. Each of these neurons consists of a cell body (*soma*), dendrites, synapses and axons. Although in the brain many specialised neurons exist, only the basics will be explained here. The neuron, see figure 1.1 receives input signals through dendrites and synapses. These inputs come from the nerves or other neurons. Such signals arrive at a synapse which determines to what extent the signals are transmitted to the dendrites. Thus the signal is amplified by the synapse in some way. This may result in excitation, a positive contribution to the dendrite's signal, or inhibition, a negative contribution. The dendrites transport the signals to the soma where they are accumulated in what is known as an action potential. When this action potential reaches a certain threshold an output spike is produced. This spike lasts about 1 millisecond and is transported along the axon to other neurons.

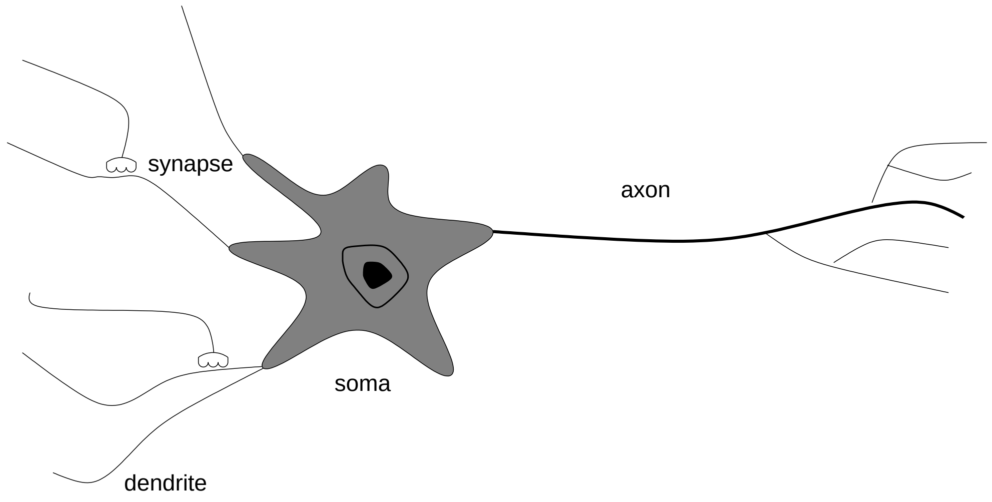


Figure 1.1: *The biological neuron.*

The brain consists of millions of these neurons, which may be specialised to some task or not. A powerful property of the brain is its ability to learn. This is assumed to be done by modifying the synaptic connections by growing new dendrites and lengthening the axon [BJ90]. By modifying the synaptic connections the brain is able to learn new tasks or to cope with a loss of function elsewhere in the brain. This behaviour of the brain inspired McCulloch and Pitts [MP43] in the early 1940s to devise an artificial neuron called the *perceptron*. This perceptron is the basis of all neural network models which have been developed. It is a simplified model of the biological neuron. Figure 1.2 shows the perceptron.

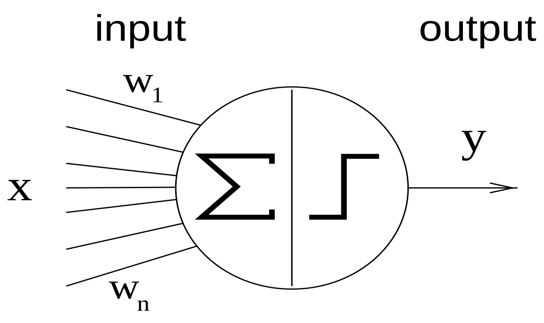


Figure 1.2: *The simplified artificial model of a biological neuron.*

The dendrites and synapses are replaced by input lines that receive inputs from other neurons or the real world. With each of these lines a weight is associated which acts like a dendrite and a synapse [BJ90]. Input signals are multiplied by the weights. The soma is modelled by a body consisting of a summation followed by a thresholding function [MP43]. This thresholding function is a step function, although in modern neural networks sigmoidal functions are usually used. Whenever the sum of input signals reaches the threshold, an output signal is given on the output line which is connected to other units.

Learning is usually done by looking at examples and extracting some general rules from them, or by using a teacher. The teacher gives examples and the brain receives rewards for correct behaviour and penalties for faulty behaviour according to the teacher's decision. For instance when children are taught to learn certain actions this is usually done by the mother/father who corrects the child for his/her action. This can also be simulated in the perceptron. Remember that the synapses have been replaced by weights, therefore by changing the weights the behaviour of the perceptron changes. Other features of the biological brain, such as the spiking phenomenon, need to be simulated using more complex models.

Changing the weights according to some rule, *the learning rule*, using a set of examples is called learning. In general two types of example sets can be distinguished: those which have for each example a desired label and those which do not. These labels indicate what type of example one is dealing with, e.g. the label "apple" indicates the example is an apple. With these labels, numbers are associated in order to be able to train a network. These numbers are referred to as *targets*. Learning rules that use examples which have a corresponding label are called *supervised*. That is a teacher indicates whether the answer of a network (*output*) corresponds to the label of an example which was expected. Using only examples and no answers is referred to as *unsupervised learning*. Learning in the perceptron is done by adapting the weights in a supervised way such that the examples supplied to the perceptron produce the correct output. Consider a given set of examples. Each of the examples is labelled either $\mathcal{A}$ or $\mathcal{B}$. A perceptron should produce a 0 when an example with label A is supplied or a 1 when an example with label B is supplied. If a perceptron produces the incorrect output, e.g. a 0 ($\equiv \mathcal{A}$) when a 1 ($\equiv \mathcal{B}$) was expected, the output value of the perceptron is subtracted from the weight values (see figure 1.2). In the case of a correct prediction the output value is added to the weight values [WH60]. By using an adaptive term when adding or subtracting only a part of the perceptron's output value is used for an update such that the next time the same example is applied the output will be correct. This type of rule is known as the Widrow-Hoff delta rule [WH60].

Using this basic building block, networks can be constructed. These constructions are usually referred to as artificial neural networks, or neural networks for short. The network structures which will be dealt with in this thesis are the so-called feed forward neural networks. This means that the input to the network is processed in a layered manner. Basically this allows a black-box approach. These networks consist of an input layer, i.e. the part of the network which “sees” the input from the real world, hidden layers and an output layer, i.e. the part of the network that produces output to the real world. The hidden layers are called hidden since they have no connection with input or output, i.e. they constitute the black-box and have no connection to the real world. These networks can quite easily be built using perceptrons. However, in the 1960s Minsky and Papert [MP69] showed in their famous book *Perceptrons* that single perceptrons have a fundamental restriction. They showed that a single perceptron cannot solve the XOR problem since it can only induce linear functions. Multi-layer structures, however, are able to solve XOR-like problems but no adequate learning rules were available at that time. Minsky and Papert’s book caused a major decrease of interest in artificial neural networks.

The interest increased significantly when the backpropagation rule was proposed by Werbos [Wer74] and later on rediscovered by Rumelhart, Hinton and Williams [RHW86]. This caused a major breakthrough since it enabled the training of the multi-layered structures. The backpropagation rule derived its name from the fact that the errors made by the network, deviations from the desired output, are propagated back through the network. This implies that the rule uses supervision since a teacher is needed to calculate the error. Section 1.4 will give a formal description of the backpropagation-trained, feed forward network. The weights of the hidden layers are adapted according to the error made by the network. Of course this is not the only network structure which has been developed throughout the years. There exist many others which also resemble certain parts of the brain or have the same functionality. One of these classes of networks is constituted by the self-organising modules [Koh89].

Various other architectures have been developed to be able to apply neural networks in specific problems. The Hopfield [Hop82] network has been developed which has been used for solving the travelling salesman problem. The analysis of this network can be done completely using well-known techniques from physics. In his analysis Hopfield introduced the notion of an energy function. Other network models are associative memory models which store and retrieve patterns by association.

The main theme in neural network research is the question of generalisation. Generalisation is the most powerful property of a neural network. How can we detect or even guarantee that a certain network under investigation will generalise? This depends on various parameters which will be addressed when we go into the details of the pattern recognition theory. Within this theory I have tried to show what problems are encountered and to what extent neural networks can help to solve these problems.

1.2 Pattern recognition

Neural networks have been successfully applied in the field of pattern recognition research. In pattern recognition one is interested in techniques to capture the human ability to recognise and classify patterns. The basic assumption in this field is that examples (from now on referred to as *samples*) can be characterised (uniquely) by a set of (relevant) measurements, called features. After measuring those particular features, a sample can be classified by inspecting the measured feature values. Consider for example the problem of distinguishing apples from pears. For both classes relevant measures can be defined, for instance shape and colour. Whenever a sample is encountered these two features can be measured in order to be able to identify it as either an apple or a pear. The classification problem is graphically illustrated in figure 1.3.

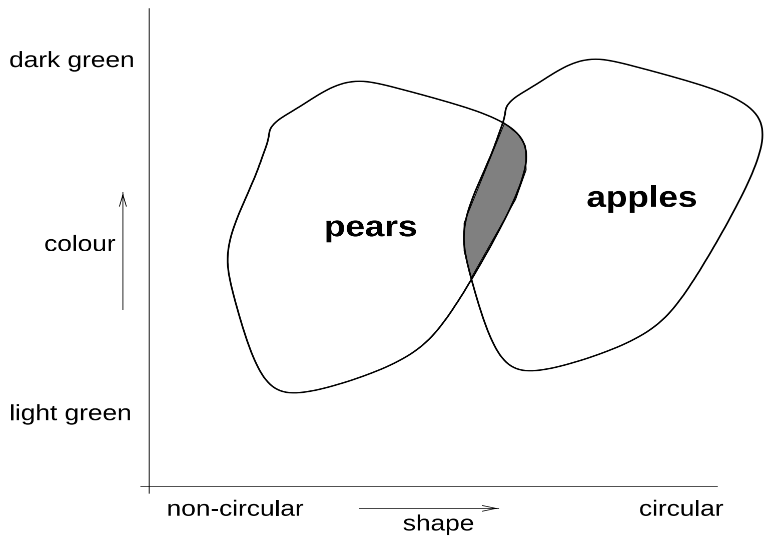


Figure 1.3: *Graphical illustration of the apples and pears problem.*

Assume that apples have roughly a circular shape whereas pears have not. Loosely speaking both sets are quite well distinguishable, there is only a small overlap between the class of pears and apples. This is due to the obvious fact that some apples and pears are lookalikes. If one would use “taste” as a third feature, no overlap would occur since apples differ from pears significantly in taste. In pattern recognition theory we are interested in how a classifier can be constructed from a set of examples that is able to separate the two classes. Moreover, the accuracy of such a classifier is of interest since many classifiers can be constructed from a limited number of samples.

In theory the optimal classifier which can be constructed, i.e. one which has the lowest possible error, is the *Bayes classifier*. Bayesian classification is based on the assumption that

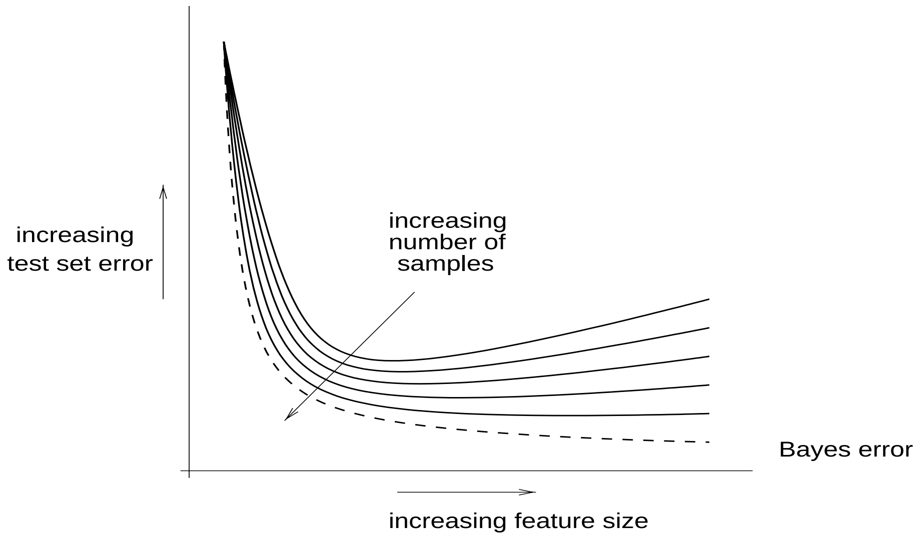


Figure 1.4: *The curse of dimensionality. For a fixed number of samples the error of a classifier will eventually increase when increasing the number of features.*

probabilities can be assigned to samples belonging to a certain class. Hence a Bayesian classifier calculates the probability that a certain class label is assigned to an observed, i.e. measured, sample. Using probabilities and density distributions assumes one either knows the correct probabilities, or one can calculate them. This constitutes a fundamental problem when applying pattern recognition in practice. In practical situations there is a need for a certain number of samples per class in order to be able to estimate the densities accurately enough [JW78, KC71]. For the precise calculation a very large number of samples is required, which one in practice not always has. Nevertheless pattern recognition algorithms have proven to be very useful in this kind of *small sample size* problem in which generalisation plays an important role. A lot of research has been done in this area [Dui95b, Rau93, JC82].

Nothing has been said yet about the construction of a classifier. In this thesis the class of *supervised* methods is used to construct classifiers. This requires a set of samples with their corresponding class labels, i.e. the class membership, that can be used for training. Classifiers that are constructed in such a way must be tested as to their correctness, i.e. the error a classifier makes after being constructed, again using a set of labelled samples. From the literature [KC71] it is known, that using the training set for testing leads to biased estimates. Hence one requires an independent test set for testing. This may cause problems in practical situations. Usually only one set of samples is available from which both a training and a test set have to be drawn. In the literature algorithms can be found which solve this problem [Bis95, Rip96]. Examples are for instance leave-one-out or bootstrapping methods. The main pattern recognition algorithms used in this thesis are: a *linear classifier*, a *quadratic classifier* and the *k nearest neighbour classifier*. They are formally described in section 1.4.

Another problem in pattern recognition practice is the number of features to be used for describing a sample. Given a limited number of samples, the number of features has to be restricted in order to be able to construct sensible classifiers. This problem is usually referred to as the *curse of dimensionality* [JW78, Dui78, vOY78]: keeping the number of samples fixed and increasing the number of available features will eventually lead to badly performing classifiers. In figure 1.4 this problem is graphically illustrated.

In figure 1.4 the lowest possible error is the theoretical Bayes error. For a fixed number of samples, denoted by the black solid line, the increase in features will initially lead to a better performance. After a certain "threshold", however, the dimensionality of the feature space is too large compared to the number of available samples. This leads to a decreasing performance of the classifier. As more samples become available, the classifier suffers less from this problem. All of the classifiers used in the thesis suffer from this problem.

Where do neural networks fit in this pattern recognition framework? What have neural networks to offer over classical pattern recognition algorithms? A reason for their appreciation is the fact that they are able to generalise from a limited number of samples. That is, they are able to cope with new situations which have not been encountered during the learning phase [HKP91, BJ90, RHW86]. During training, neural networks are expected to induce a relationship that interpolates between or extrapolates from the training samples. So far nothing is new, classifiers from pattern recognition theory are also said to have such properties. The remarkable thing about neural networks, however, is that neural network architectures have in general a huge number of adjustable parameters compared to traditional classifiers. For example in the famous *Nettalk* [SR86] application only 5,000 samples were used to determine 25,000 network parameters. From a pattern recognition point of view this is hardly achievable. What happens inside a neural network that enables this? Research has shown, that due to the methods used for finding the appropriate parameter values the number of effective parameters is much smaller [Moo92, Kra93].

During training there is a situation in which a network optimally generalises. Before this occurs both the training error and the test set error can still be improved, i.e. they are still decreasing. They approach a situation in which the network starts to generalise (see figure 1.5). Beyond this generalisation situation, the training error still decreases (or is negligible) and the test set error rises again (figure 1.5). When this occurs, a network is said to be in an *overtrained* situation. In such situations the generalisation of a network deteriorates. The aim of a training procedure is therefore to find the optimal trade-off between training error and test set error. Figure 1.5 depicts a typical training situation of a neural network.

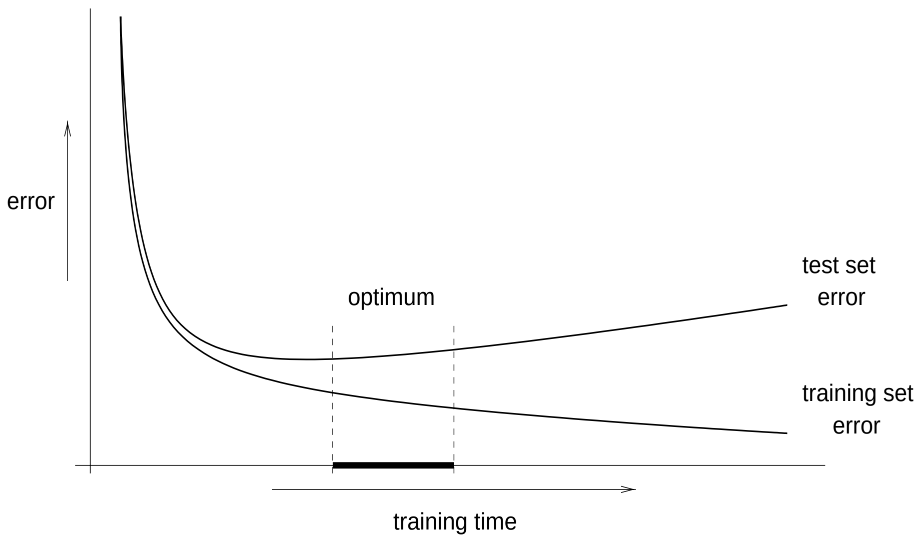


Figure 1.5: *A typical training situation for a neural network.*

A major problem in current neural network research is therefore to find the appropriate moment to stop training. This requires a thorough understanding of the working of a neural network. The knowledge is already partially available in the current literature [HKP91, Bis95, Rip96]. This thesis aims to extend the knowledge of the working of neural networks by introducing and evaluating new tools in the analysis of neural network generalisation behaviour. Furthermore, the relationship between neural networks and classical pattern recognition methods is emphasised concerning their generalisation behaviour.

1.3 Overview of the thesis

The general theme of this thesis is the generalisation capability of neural network classifiers. Specifically we are dealing with the class of feed forward, backpropagation-trained neural network. We are interested in whether one is able to detect and maybe guarantee generalisation in these classifiers and relate their behaviour to the “classical” ones such as the k nearest neighbour classifier. This classifier is the most natural one to use in statistical pattern recognition [MST94]. The subsequent chapters will deal with the theory and tools that can be used for studying the classifier behaviour.

The remainder of **chapter 1** is devoted to the formal introduction of the general methods from neural network theory and pattern recognition. It also introduces some nomenclature which is used throughout the thesis and which is supposed to be generally known. Any specific notations will be introduced in the chapters themselves.

Chapter 2 studies the current theory of generalisation to provide a theoretical framework for generalisation. It introduces two more or less opposite viewpoints to generalisation in classifiers, the *Bayesian* and *Uniformist*. These two theories are studied and discussed on their merits. Since these two theories are worst case approaches to the problem of generalisation another view is adopted. This view is known as the *compactness assumption* and leads to some properties which are assumed to be unknown in the two theories. The last subsection of this chapter introduces the *Support Vector Classifier* (SVC) which is said to be a classifier which has the best generalisation possible given the data.

The next chapter, **chapter 3**, is concerned with the monitoring of the training behaviour of classifiers. As stated in section 1.1 there is some "optimal" region in which the classifier is generalising. During training this region needs to be detected in order to find a good classifier. This chapter introduces the notion of *nonlinearity* of classifiers to study their behaviour. It is shown that it is a useful tool in generalisation studies for both neural classifiers and classical pattern recognition classifiers.

The choice of the architecture, e.g. the number of hidden units, is an important problem in neural network research. Taking a large number of hidden units helps in neural network learning. **Chapter 4** studies the behaviour of the neural network when dealing with *redundancy*, i.e. a superfluous number of hidden units. Methods are introduced with which this redundancy can be detected and shown. Furthermore it is shown how the learning algorithm influences redundancy by comparing three learning rules.

In order to judge the reliability of classifications made by a classifier, **chapter 5** introduces **our** notion of *confidence values*. These values can be used to determine the reliability of a sample classification made by the network. These reliabilities can be used for *rejection* to improve the quality of the classifier.

Chapter 6 will discuss how the methods introduced in the previous chapters behave in practical applications. In each of the previous chapters the methods were studied using a controlled environment in order to investigate their behaviour. Chapter 6 studies the well-known problem of handwritten digit recognition which is a commonly used benchmark in neural network research [LCBD$^+$89, WG90]. It describes how the various tools behave in a practical situation.

Finally the thesis ends with **chapter 7** which contains the conclusions regarding the research described in the thesis. Although each of the individual chapters has its own discussion regarding the material presented in that specific chapter, chapter 7 presents and discusses some topics regarding neural network research in general.

1.4 Formalisms used in the thesis

In this section the concepts introduced in the previous sections will be dealt with in a formal way. As mentioned before, a sample can be characterised by its measurements or features. A sample, denoted by $\vec{x}$, can be written as a vector of feature values $\vec{x} = \{x_1, x_2, \ldots, x_n\}$. When a classifier is used, these feature values serve as its input. Classifiers are denoted by $\mathcal{S}$ and represent both "classical" classifiers and neural network classifiers. These classifiers assign a label, denoted by ω. This label ω represents the class to which the sample $\vec{x}$ is assigned. In general we have a set of m possible class labels $\mathcal{W} = \{\omega_1, \omega_2, \ldots, \omega_m\}$. Formally a classifier performs the mapping $\mathcal{S} : \vec{x} \to \omega_i$, or $\mathcal{S}(\vec{x})$ for short.

Classifier training is done using three types of data sets. These are denoted by $\mathcal{L}$, the *training* set, $\mathcal{T}$, the *test* set and finally $\mathcal{V}$, the *validation* set. Each of these sets has the following form: $\{(\vec{x}_1, \omega_1), (\vec{x}_2, \omega_2), \ldots, (\vec{x}_n, \omega_n)\}$, where $n \in I\!N$ is some whole number larger than 0. The sets may have different sizes, but all have the same form of being a set of pairs, a sample and its corresponding label. Training sets are used to train a network, whereas the test set is meant as an independent set to judge the quality of the network. A validation set can be used for instance during training as stopping criterion. This cannot be done using a training set since this results in biased classifiers.

In the Bayesian framework, probabilities are used. These are denoted by $P(event)$, i.e. the probability that a certain event occurs. $P(event2|event1)$ is also used, indicating the probability of $event2$ when having observed $event1$. The Bayes rule is defined as follows:

$$P(a|b) \;=\; \frac{P(b|a)P(a)}{P(b)}, \tag{1.1}$$

This rule enables the definition of the Bayes classifier. Let $\mathcal{S}$ be a classifier that assigns an $\vec{x}$ to class A iff $\mathcal{S}(\vec{x}) \geq 0$ and to class B iff $\mathcal{S}(\vec{x}) < 0$. Furthermore, let the apriori probabilities for both classes be denoted by P_A and P_B, and their probability density functions by $P(\vec{x}|A)$ and $P(\vec{x}|B)$. The error ε made by this classifier is given by:

$$\varepsilon \;=\; P_B \int_{\mathcal{S}(\vec{x})=\omega_A} P(\vec{x}|B)d\vec{x} + P_A \int_{\mathcal{S}(\vec{x})=\omega_B} P(\vec{x}|A)d\vec{x}. \tag{1.2}$$

The error is given by the possible misclassifications made by the classifier. If $\mathcal{S}$ minimises ε (which can be done by using the Bayes rule) then it is said to be the Bayes classifier and the Bayes error is often denoted by ε_*. It is the lowest achievable error given the class density distributions and a priori probabilities. In general, however, only $\mathcal{L}$ is available. From this $\mathcal{L}$ the classifier has to be constructed and, if necessary, the probabilities. The construction of $\mathcal{S}$ from a set of samples $\mathcal{L}$ is usually referred to as *learning from examples* and is done using a *learning algorithm*. The next subsections show some learning algorithms which are frequently used in this thesis. After the construction of $\mathcal{S}$ its error can be calculated on $\mathcal{L}$, called the *learning error*, which is denoted by ε_a (the apparent error). The error on the test set $\mathcal{T}$ is denoted by ε_t. Both errors are usually defined as the percentage of misclassified samples, i.e. the number of misclassified samples divided by the total number of samples. The aim of the learning algorithm is to find the trade-off between ε_a and ε_t such that both errors are as low as possible. This leads to the definition of generalisation in (neural) classifiers. In this thesis the following definition will be used:

Definition 1 (*Generalisation***)** :
Generalisation is the ability of the classifier $\mathcal{S}$ to correctly classify samples which have not been encountered during training. The expected error made on such samples is the generalisation ability, or generalisation, of a classifier $\mathcal{S}$. □

Wherever generalisation is mentioned in the thesis, we refer to this concept.

1.4.1 Neural network algorithms

The most basic building block in the neural network is the perceptron. It is characterised by several input lines with weights associated to them. The input is collected and summed and an output is given according to the output function f. The output is formally defined as follows:

$$o = f\left(\sum_{i=1}^{n} w_i \vec{x}_i\right), \tag{1.3}$$

where o is the output function and w_i the weight associated with input line i. The McCulloch-Pitts neuron has a step function as output function (see figure 1.2). This neuron can be trained using the Widrow-Hoff delta rule [WH60]. Assume the feature values of the samples are real numbers, and their labels are either 0 or 1 denoting two different classes. The delta rule adapts the weights w_i according to the following formula:

$$w_i(t+1) = w_i(t) + \eta(y_i(t) - o(t))\vec{x}_i(t), \tag{1.4}$$

where $y_i(t)$ is the desired response, i.e. the real class label, and η an adjustable parameter called the *learning rate* which usually has a value between 0 and 1. The t is used to indicate the time step. Note that there is no change in the weights if the perceptron produces a correct answer, $y(t) - o(t) = 0$, in other cases the weights are adapted. As said in section 1.1, Minsky and Papert showed the limitation of this perceptron. It can only implement linear decision functions. This makes it unsuited for solving the XOR problem, which need a non-linear solution. It was theoretically shown that it cannot be solved by a single perceptron but only by constructing a layered architecture [MP69]. This layered architecture, depicted in figure 1.6, has the problem that it cannot be trained using the standard delta rule, since it does not allow the update of the weights from a hidden layer to the input layer. This update requires an error which cannot be calculated.

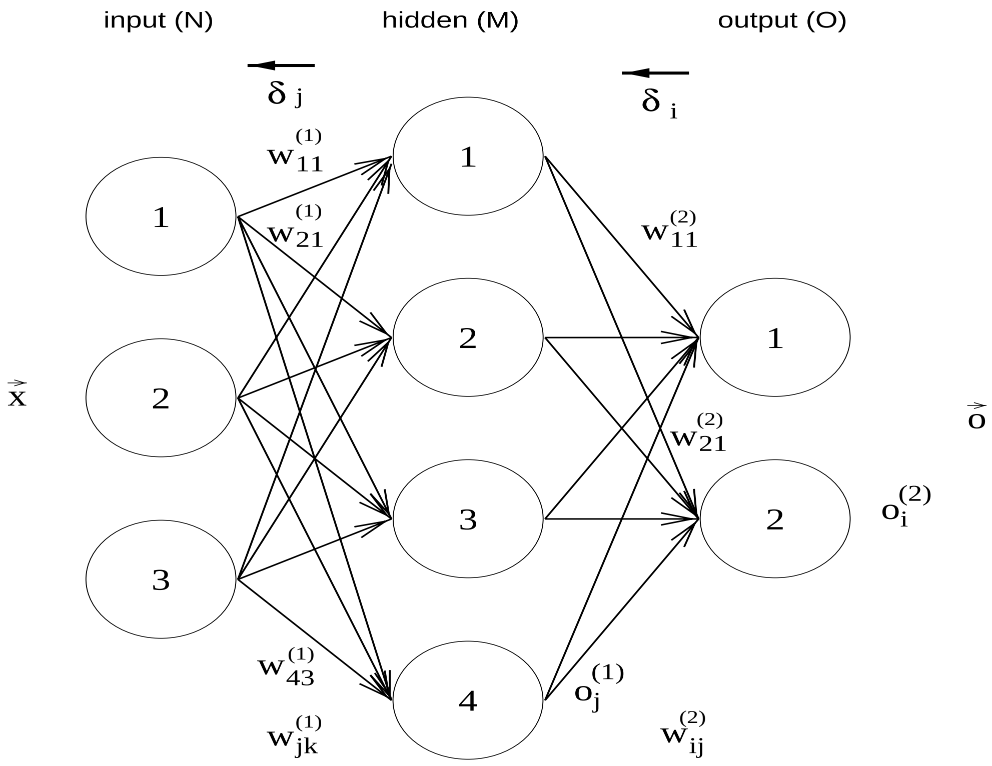


Figure 1.6: *A layered structure of perceptrons.*

The invention of the backpropagation rule caused a major breakthrough in neural network research. In principle the rule is very simple: calculate the error made by the network and propagate it back through the network layers. This back-propagated error is used to update the weights. Consider figure 1.6. On the left side the sample $\vec{x}$ is fed to the network and produces an output on the right side $\vec{o}$. The input pattern $\vec{x}$ is propagated through the network in the following way:

$$o_j^{(1)} = f\left(\sum_{k=1}^{N} w_{jk}^{(1)} \vec{x}_k\right), \tag{1.5}$$

$$o_i^{(2)} = f\left(\sum_{j=1}^{M} w_{ij}^{(2)} o_j^{(1)}\right), \tag{1.6}$$

where $o_j^{(1)}$ and $o_i^{(2)}$ denote the output of a hidden unit and an output unit respectively. The variables N and M denote the number of input units and the number of hidden units. A weight from a unit to another unit is denoted by $w_{ij}^{(l)}$ where j is the "source" of the connection, i the "target" and l the layer. The final output of the network can be written as:

$$o_i^{(2)} = f\left(\sum_{j=1}^{M} w_{ij}^{(2)} f\left(\sum_{k=1}^{N} w_{jk}^{(1)} \vec{x}_k\right)\right). \tag{1.7}$$

where $o_j^{(1)}$ has been replaced by equation 1.5. The output of the network has to be judged using some error criterion. This criterion determines the size of the error to be back propagated. In general the mean squared error (MSE) criterion is used:

$$E = \frac{1}{2}\frac{1}{|\mathcal{L}|}\sum_{i=1}^{|\mathcal{L}|}\sum_{\vec{x}\in\mathcal{L}}[y(\vec{x}) - o(\vec{x})]^2, \tag{1.8}$$

where $y(\vec{x})$ is the desired network output value for the sample $\vec{x}$ under investigation and $|\mathcal{L}|$ the size (cardinality) of the learning set. The objective during training is to minimise this error by choosing the appropriate weights. By differentiating equation 1.8, the desired change can be calculated to minimise E. This requires f to be differentiable. This cannot be done when using the step function, as differentiating leads to a delta (impulse) function. A step function, however, can be approximated by a sigmoidal function:

$$f(z) = \frac{1}{1 + \exp(-z),} \tag{1.9}$$

If the weights of a unit are very large $f(z)$ will approximate the step function (see equation 1.3 where z is the weighted sum). Furthermore the sigmoid function has the property that its derivative can be expressed in terms of itself: $f'(z) = f(z)(1 - f(z))$. This means that one can easily calculate the derivative. The weights of the neural network are updated using a gradient descent procedure [Sim90, HKP91, BJ90]. First the error of the output units is calculated. The change in weight for output unit i from hidden unit j is given by the gradient:

$$\begin{aligned}\Delta w_{ij}^{(2)} &= \frac{\partial E}{\partial w_{ij}^{(2)}} \\ &= \sum_{\vec{x}\in\mathcal{L}}[y_i - o_i^{(2)}]f'(o_i^{(2)})o_j^{(1)}.\end{aligned} \tag{1.10}$$

By using the chain rule the weight change for the hidden layer can be computed. The weight change for hidden units is given, using the chain rule [Wer74, RHW86, HKP91, BJ90], by:

$$\begin{aligned}
\Delta w_{jk}^{(1)} &= \sum_{\vec{x}\in\mathcal{L}} \frac{\partial E}{\partial o_j^{(1)}} \frac{\partial o_j^{(1)}}{\partial w_{jk}^{(1)}}, \\
&= \sum_{\vec{x}\in\mathcal{L}} \sum_i [y_i - o_i^{(2)}] f'(o_i^{(2)}) w_{ij}^{(2)} f'(o_j^{(1)}) \vec{x}_k, \\
&= \sum_{\vec{x}\in\mathcal{L}} \sum_i \delta_i^{(2)} w_{ij}^{(2)} f'(o_j^{(1)}) \vec{x}_k, \\
&= \sum_{\vec{x}\in\mathcal{L}} \delta_j^{(1)} \vec{x}_k, \qquad (1.11)
\end{aligned}$$

where $w_{jk}^{(1)}$ is the weight from hidden unit j to input unit k (see figure 1.6 for the graphical details). The $\delta_i^{(2)}$ is given by $\delta_i^{(2)} = f'(o_i^{(2)})[y_i - o_i^{(2)}]$, the error made in the output layer. This δ is back propagated to the hidden layers, hence the term *backpropagation*. In the term $\delta_j^{(1)} = f'(o_j^{(1)}) \sum_i w_{ij}^{(2)} \delta_i^{(2)}$ this δ_i from the output is used. Therefore, to compute the weight changes in a hidden layer only the δ from an upper layer needs to be known.

The new weight value is determined by the learning algorithm. In its most simple form the new weight is determined by:

$$w_{new} = w_{old} - \eta \Delta w. \qquad (1.12)$$

Note that the $-$ sign is due to the fact that $\Delta w < 0$ since a gradient direction is calculated. This update rule, however, has some limitations regarding convergence. Therefore the update rule with *momentum term* α ($0 \leq \alpha < 1$) is used combined with a more sophisticated scheme to adaptively control the update [HKP91]. This leads to the following rule:

$$w_{new} = w_{old} - \eta \Delta w - \alpha w_{old}. \qquad (1.13)$$

Another way to calculate the derivative is to use second order information. The gradient descent method in equation 1.10 only uses the first order term in the derivative of the error. By taking a Taylor-series expansion of the error function around the current point $\vec{x}_0$, followed by taking the derivative one obtains:

$$\begin{aligned}
E(\vec{x}) &= E_0 + (\vec{x} - \vec{x}_0) \cdot \nabla E(\vec{x}_0) + \frac{1}{2} (\vec{x} - \vec{x}_0) \cdot \mathbf{H} \cdot (\vec{x} - \vec{x}_0) + \ldots, \\
\nabla E(\vec{x}) &= \nabla E(\vec{x}_0) + \mathbf{H} \cdot (\vec{x} - \vec{x}_0) + \ldots, \qquad (1.14)
\end{aligned}$$

where $\mathbf{H}$ is the second derivative Hessian matrix. Using this Hessian matrix causes the learning algorithm to converge in a much faster way, It is, however, computationally expensive to compute it and sometimes even impossible for large neural networks. In this thesis we use the well-known Levenberg-Marquardt [Lev44] (LM) rule for the estimation of the Hessian. The approximation of the Hessian uses outer products and is given by:

$$\begin{aligned} \mathbf{H}_{ij} &= \frac{\partial^2 E}{\partial w_{ij} \partial w_{jk}} \\ &= \sum_{\vec{x} \in L} \frac{\partial o^{\vec{x}}}{\partial w_{ij}^{(2)}} \frac{\partial o^{\vec{x}}}{\partial w_{jk}^{(1)}}, \end{aligned} \tag{1.15}$$

where $o^{\vec{x}}$ is the output of the network for sample $\vec{x}$. The learning rule shown in equation 1.13 changes into:

$$w_{new} = w_{old} - (\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{Z}^T \epsilon(w_{old}), \tag{1.16}$$

where $\mathbf{Z}_i \equiv \partial \vec{x}_k / \partial w_i$ [Bis95] and $\epsilon(w_{old})$ the error on the previous weights. This learning rule is also used in the thesis next to the standard rule with momentum.

1.4.2 Pattern recognition algorithms

There exists a wide variety of pattern recognition algorithms. In this subsection the algorithms used in the thesis will be formally introduced. Two types of algorithms are used, linear and nonlinear ones. The linear algorithms are only capable of generating linear classification functions, whereas nonlinear ones can generate in principle any kind of classification function. The algorithms used are:

- Linear classifier.
- Quadratic classifier.
- k Nearest neighbour classifier (*k*-NN).

Each of these classifiers will now be explained in more detail. Consider the example of the apples and pears depicted in figure 1.3. In order to separate the two classes one needs a decision function and it should be constructed in such a way that the error is as small as possible.

A classifier which can be constructed for the two classes is the following:

$$\mathcal{S}(\vec{x}) = (\vec{x} - \mu_A)^T \mathbf{G}_A^{-1} (\vec{x} - \mu_A) - (\vec{x} - \mu_B)^T \mathbf{G}_B^{-1} (\vec{x} - \mu_B) + \ln \frac{|\mathbf{G}_B|}{|\mathbf{G}_A|}, \tag{1.17}$$

where $\mathbf{G}$ is the covariance matrix for each class, and μ the mean of each class. However, this classifier can only be calculated when the distributions are known. In practical situations estimates of $\mathbf{G}$ and μ have to be calculated. Furthermore, the classifier stated in equation 1.17 is a quadratic one. The linear classifier, however, can easily be derived as a special case. If both classes have equal covariances ($\mathbf{G}_A = \mathbf{G}_B$) equation 1.17 reduces to a linear classifier. The problem of estimating μ and $\mathbf{G}$ can be solved by estimating them from the learning set $\mathcal{L}$. The mean of a class can be estimated using the average over the feature values of the samples belonging to that particular class, i.e.:

$$\hat{\mu}_A = \frac{1}{N_A} \sum_{\vec{x} \in \mathcal{L}_A} \vec{x}, \tag{1.18}$$

$$\hat{\mu}_B = \frac{1}{N_B} \sum_{\vec{x} \in \mathcal{L}_B} \vec{x}, \tag{1.19}$$

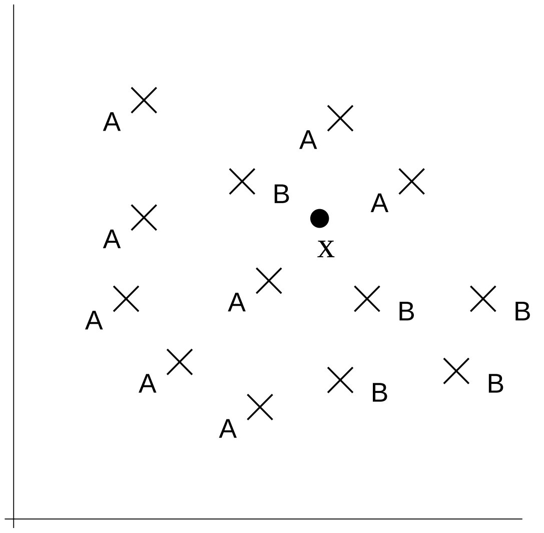


Figure 1.7: *Example of the k nearest neighbour rule.*

The estimation of the covariance matrix is somewhat more difficult, especially since a matrix inversion is later required. This requires the number of samples in $\mathcal{L}$ to be at least as large as the dimensionality of the features, i.e. the matrix $\mathbf{G}$ is required to be non-degenerate. The covariances can be estimated by:

$$\widehat{G}_A = \frac{1}{N_{\mathcal{A}} - 1} \sum_{\vec{x} \in \mathcal{L}^A} (\vec{x}_i - \hat{\mu}_A)(\vec{x}_i - \hat{\mu}_A)^T, \tag{1.20}$$

$$\widehat{G}_B = \frac{1}{N_{\mathcal{B}} - 1} \sum_{\vec{x} \in \mathcal{L}^B} (\vec{x}_i - \hat{\mu}_B)(\vec{x}_i - \hat{\mu}_B)^T, \tag{1.21}$$

where $\mathcal{L}$ is divided in the two sets $\mathcal{L}^A$ (every $\vec{x}$ belonging to this set has label $\mathcal{A}$) and $\mathcal{L}^B$. Note that $\mathcal{L} = \mathcal{L}^A \cup \mathcal{L}^B$ and $\mathcal{L}^A \cap \mathcal{L}^B = \emptyset$. The linear and quadratic classifier have been investigated quite thoroughly by many people and usually serve as a reference for other classifiers (at least in this thesis). The aforementioned perceptron bears a close resemblance to the linear classifier.

The next classifier that is used in the thesis, is the k nearest neighbour classifier. Consider the set of samples in figure 1.7. Each of these samples ($\times$) has been labelled either A or B, except for one sample (•). This sample needs to be labelled using a classification function. The k nearest neighbour classifier takes the k nearest, i.e. closest, neighbours around the sample $\vec{x}$ and uses them to assign a label. This is usually done by a majority voting rule which states that the label to be assigned should be the one which occurs the most among the neighbours. At least two problems arise when using this classifier. The first problem is to find a suitable distance measure that tells which sample ($\times$) is closer to the sample (•). The second problem is the choice of *k*: choosing *k* large generally results in a linear classifier whereas small *k* result in nonlinear ones. This influences the generalisation capability of the *k*-NN classifier.

The distance metric used to calculate distances between samples is usually the Euclidean distance. Given two samples $\vec{x}$ and $\vec{y}$, the Euclidean distance ($||\vec{x} - \vec{y}||$) between the two samples is defined as:

$$||\vec{x} - \vec{y}|| \quad = \quad \sqrt{\sum_{i=1}^{n} (\vec{x}_i - \vec{y}_i)^2}, \tag{1.22}$$

where n the number of features describing $\vec{x}$ and $\vec{y}$. Another distance measure is for instance the city-block distance, which is defined as:

$$D_{city-block}(\vec{x}, \vec{y}) \quad = \quad \sum_{i=1}^{n} |\vec{x}_i - \vec{y}|. \tag{1.23}$$

Choosing a correct k is a much harder problem. Too large (or too small) k may result in non generalising classifiers. The optimal k can be found by using for instance the leave-one-out method on the training set. This of course still requires independent test sets for accurate error estimation and comparison of different *k* nearest neighbour classifiers.

Of course there are many other pattern classification algorithms available in the literature. The interested reader is referred to for instance [Fuk90, DH73, Rip96, Bis95].

Chapter 2

Generalisation

2.1 Introduction

Generalisation is an important property of classifiers. With their ability to generalise to unseen samples it is possible to train classifiers on finite, sometimes even small, data sets. In this chapter we show how generalisation is dealt with in the existing literature. This will be done by presenting two frameworks which we consider to be important. These frameworks are most commonly used, although there do exist others. We will go into the details of the following two frameworks:

1. Bayesian framework, especially the extended Bayesian framework introduced by Wolpert [Wol93b, Wol93a, Wol94].

2. Uniform convergence framework of Vapnik [Vap82, Vap95].

These two frameworks constitute two different views towards the analysis of classifier behaviour. The Bayesian framework is mainly concerned with the analysis of probabilities associated to classifiers. By assuming some prior distribution on the possible classifiers, the posterior probabilities can be calculated using the Bayes rule and the given data. These posterior probabilities are used to test whether the prior assumptions were correct, i.e. if the posterior approximates the target, i.e. the best possible classifier, adequately one has chosen the right prior.

The uniform convergence framework is not based on any distributions. It is concerned with an upper bound on the number of samples needed to reach a certain error related to the classifier under investigation. Basically the only thing that is important in this framework is the size of the data set and the *capacity* of the classifier. The capacity of a classifier is defined as the number of dichotomies it can induce on an arbitrary data set of a certain size (see section 2.2.2 for more details). This is important since it tells something about the generalisation capability of the classifier.

The way these frameworks are described here is rather coarse. In practice the differences are not always that clear, especially when used in real-life applications. We will adopt another perspective with respect to the classifier and the data it has to classify, by using the compactness assumption of the data [AB66]. Basically this implies that the data cannot be arbitrarily scattered in the feature space. This has implications for both the a priori assumptions of the data distribution and the classifier training and consequently the generalisation behaviour. Since in general our data is obtained from real-life measurements, the samples do not represent a single point but a small neighbourhood due to the error or natural variations in the measurements of the samples. In principle we are only interested in the overlapping part of the data. That is the part of the data where the classes intersect. It may be the case that the classes are perfectly separable or that classes have a real overlap as in the apples and pears problem (figure 1.3). In both cases, however, we are interested in the class borders. This is not taken into account when the uniform convergence framework is used. The support vector classifier, however, has been developed (see section 2.3) which considers the touching part of the data.

The remainder of this chapter is divided as follows. Section 2.2 starts with the introduction of Wolpert's viewpoint followed by the explanation of Vapnik's uniform convergence framework. The last part of the section is devoted to the introduction of the compactness assumption. The following section (2.3) introduces the support vector classifier, which yields a classifier having an optimal generalisation capability with respect to the data. The chapter will end with a discussion.

2.2 Theoretical analysis of generalisation

In the literature several ways to investigate generalisation behaviour of neural network classifiers have been proposed. Basically there are two viewpoints. The first viewpoint, as proposed by Wolpert, considers the probability of having found a classifier that matches the one that one is actually seeking, given the data distribution. The other viewpoint, of Vapnik, is interested in the possible classifiers and the number of available samples and does not make any assumptions about the data distribution nor on the distribution of the possible classifiers. Since no distribution assumption is made it is more important than the Bayesian framework. In this section both viewpoints are described. However, these viewpoints are too pessimistic for practical use since they result in too loose bounds and therefore the use of the compactness in the classification problem will be discussed.

2.2.1 Bayesian framework

Wolpert considers the generalisation problem from the Bayesian framework. This implies that he is only interested in the probabilities of occurrences. Moreover, by using the Extended Bayesian Framework (EBF) [Wol93b] he shows that other frameworks, such as Vapnik's [Vap82], can be incorporated. In principle Wolpert does not make any prior assumption on the distribution of the data and classifier. It can have any distribution, which is unlikely in applications based on physical measurements. This will be shown later on. This poses a problem: If any distribution of the data and classifier is possible then on the average (over all problems) any learning algorithm will have the same performance. This was shown by Wolpert in his no-free-lunch (NFL) theorem [Wol93b, Wol93a, Wol94] and practically investigated by Zhu and Rohwer [ZR95]. Wolpert states that no a priori assumption can be made since no information is available on the distributions of the classifier and the data. Any prior assumption is therefore false and may lead to a wrong prediction. He shows that it is impossible to justify a correlation between reproduction of a training set and generalisation error of the training set using only a priori reasoning. However, Wolpert considers the off-training set generalisation error, which is an error induced by the samples not belonging to the training set. In fact these samples may have any label as long as they were not part of any training set used to find the classifier. We are interested in problems in which the training set and test set share the same characteristics and are based on the same distributions. If the test set is allowed to be any set, it is hard to draw any conclusions about the validity of the classifier found by the learning algorithm.

Extended Bayesian framework

The extended Bayes formalism used by Wolpert [Wol94] is the classical probability theory applied to the case where two different variables are used for the classifier $\mathcal{S}$ found (the hypothesis) and for the classifier target (target relationship f), i.e. the classifier that would solve the classification problem best. Traditionally the Bayesian approach is the following:

- Define a model space, a space which one believes to contain the "true" model.
- Assign prior probabilities to the models and probability distributions to the values of the model parameters. In the Bayesian framework $P(\mathcal{S}|d)$ (the probability of finding the classifier $\mathcal{S}$ given the data) is usually fixed. Wolpert states that this is not the case in his framework.
- Use the Bayes rule to find the a posteriori probabilities of the models given the data.
- Choose a search algorithm which searches the space of models to find the one(s) having the maximum posteriori probability.
- Stop when an appropriate model has been found according to the stopping criterion.

Wolpert's approach differs from the standard one in that it assumes a uniform prior of target functions (the hypothetical classifier that assigns the correct class labels) and not a single model. The procedure according to Wolpert is the following. First an event space is defined, consisting of "target functions" (denoted by f) that are the functions one wants to approximate using the classifier. Furthermore there are "hypothesis functions" which are the result of training, e.g. a classifier $\mathcal{S}$ and a training set denoted by d. The training set is defined as a set of sample and label pairs $(\vec{x}, \omega)$. Target functions denote the actual relation between inputs x and labels ω. The output generated by a search algorithm is the classifier $\mathcal{S}$ or hypothesis function, the approximation of f. The probability $P(f)$ is the probability of a particular target function, this value is given by the environment. As opposed to the Bayesian approach where one makes an estimate of this probability, Wolpert makes none. This is a fundamental difference since Wolpert states that one cannot make any assumptions since it is outside the control of the researcher [Wol93b].

Supervised learning consists of fixing aspects of the probability $P(f, \mathcal{S}, d)$, note that $P(f) = \int_{\mathcal{S},d} P(f, \mathcal{S}, d)$. For instance one can fix the training set d and calculate the probability by averaging over different classifiers. Furthermore, with a certain choice for $\mathcal{S}$ and / or d a cost c is associated, $P(c|f, \mathcal{S}, d)$. Usually this cost is given by a loss function L and is chosen to be the squared error, $L \equiv (\mathcal{S}(\vec{x}) - f(\vec{x}))^2$, or the error count, $L \equiv \int \pi(\vec{x})[1 - \delta(f(\vec{x}), \mathcal{S}(\vec{x}))]$. Here $\pi(\vec{x})$ is the sampling distribution and $\delta(\vec{x})$ the delta function which gives a 1 iff $\vec{x} \equiv \vec{0}$. A learning algorithm usually computes the probability $P(\mathcal{S}|d)$, i.e. the probability of a certain hypothesis given the training set. So Bayesians are not directly interested in the training of a classifier but in the probability that given their prior they can compute the right posterior.

Generalisation is now considered as the probability of agreement between $P(f|d)$ and $P(\mathcal{S}|d)$. The larger this probability the better generaliser one has found. The problem however is that in the framework it is not stated how a classifier is obtained. It can be done by pure guessing or by something that is only slightly better. This is what lead to the no-free-lunch theorem. This theorem states that for any two algorithms $\mathcal{A}$ and $\mathcal{B}$, there are as many targets for which $\mathcal{A}$ has a lower generalisation error than $\mathcal{B}$ as vice-versa. Note that it does not matter whether one averages over training sets or not. Zhu and Rohwer [ZR95] have shown that the NFL theorem holds for the cross validation method. In real life experiments, however, one has prior knowledge which can be used. This implies that in our *compactness assumption* (for details see section 2.2.3) the no-free-lunch theorem does not hold since we have prior knowledge about how data is obtained.

2.2.2 Uniform convergence framework

Vapnik considers the problem of generalisation from the viewpoint of the classifier under investigation. He derives a bound on the number of samples needed to guarantee a certain error for a certain class of classifiers under investigation. The framework in which this is calculated is referred to as the uniform convergence framework.

In principle Vapnik's bounds only depend on the classifier used. For that classifier its capacity can be estimated, called the Vapnik-Chervonenkis (VC) dimension. This capacity is independent of the distribution of the samples (note that Wolpert would assume an arbitrary classifier). The problem using Vapnik's approach is that it results in an upper bound on a set of classifiers and not a single classifier. Moreover, Vapnik uses a "worst case" approach which will hardly ever be encountered in practice.

Vapnik considers learning through examples as the following [Vap95]:

- A generator which produces random vectors $\vec{x}_i$, i.e. training set, according to some unknown distribution $P(\vec{x})$. Note that Bayesians use some prior on this distribution $P(\vec{x})$.
- A supervisor which returns an output value ω for an input $\vec{x}$.
- A learning machine capable of implementing a set of functions $f(\vec{x}, \alpha)$, where α is some set of parameters for the function.

More formally, let a probability measure $P(\vec{x})$ be defined on some space $\mathcal{X}$, this probability is in general unknown, but a sampling of the probability is given by $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_l$. Let a set of functions $Q(x, \alpha)$ also be defined, i.e. functions that depend on the probability distribution and some parameter α. The goal is to find that function that minimises the error (called risk $R(\alpha)$ by Vapnik) given by:

$$R(\alpha) \quad = \quad \int Q(\vec{x}, \alpha) dP(\vec{x}). \tag{2.1}$$

In general only a limited number of samples following $P(\vec{x})$ is given. Therefore the learning problem reduces to minimising the empirical risk:

$$R_{emp}(\alpha) \quad = \quad \frac{1}{l} \sum_{i=1}^{l} Q(\vec{x}_i, \alpha). \tag{2.2}$$

The real risk $R(\alpha)$ is estimated using a learning set consisting of l samples. If the number of samples is infinite then $R_{emp}(\alpha)$ will approximate $R(\alpha)$, see for instance figure 2.1. Vapnik shows how many samples are needed given a certain classifier to bound the difference between $R(\alpha)$ and $R_{emp}(\alpha)$. The process of estimating $Q(\vec{x}, \alpha)$ using equation 2.2 is called

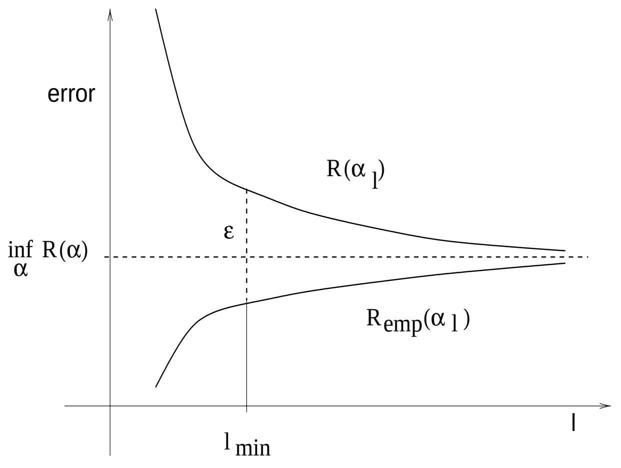


Figure 2.1: *The error curves for increasing sample sizes. $R(\alpha_l)$ denotes the expected true error (risk), whereas the $R_{emp}(\alpha_l)$ denotes the empirical error, l_{min} shows the minimum number of samples required to guarantee an expected difference of at least ε.*

empirical risk minimisation (ERM). Traditionally the estimator of the empirical risk, or empirical error ε_a, is chosen to be the mean squared error (MSE), although any loss function could be chosen:

$$R_{emp}(\alpha) = \frac{1}{l}\sum_{i=1}^{l}(y_i - f(\vec{x}_i, \alpha))^2, \tag{2.3}$$

where y_i is the desired output and $f(\vec{x}_i, \alpha)$ the output delivered by the classifier, for instance a neural network. The risk is then minimised by training the network using the training set. This principle of risk minimisation is quite general but plays a basic role in Vapnik's learning theory.

Therefore, according to Vapnik [Vap95], a learning theory should address the following questions:

1. What are the (necessary and sufficient) conditions for consistency of a learning process based on ERM? The framework for this will be the uniform convergence framework.
2. How fast is the rate of convergence of the learning process? This will be addressed by the introduction of a growth function for a classifier.
3. How can one control the rate of convergence (the generalisation ability) of the learning process? This is included in the growth function and the capacity of a classifier.
4. How can one construct algorithms that can control generalisation? This is addressed in a next section where the support vector classifier will be introduced.

The uniform convergence framework is used to ensure that the deviation of the empirical risk, i.e. the error made by the estimation, deviates from the real risk within a certain bound with a certain probability. In figure 2.1 this is graphically illustrated. If the learning process is consistent then both risks are expected to converge to the same asymptotic value. The uniform convergence theorem is defined as follows:

$$\lim_{l \to \infty} P\left(\sup_{\alpha} |R(\alpha) - R_{emp}(\alpha)| > \varepsilon\right) = 0 \quad \forall \varepsilon > 0. \tag{2.4}$$

If the learning process fulfils this equation, it is called consistent. However, one would also like to have a bound on the rate of convergence, i.e. to state that with a certain probability a certain bound is reached. Suppose one wants to know how many samples one needs to reach the bound ε in figure 2.1 with a certain probability. The upper bound on the uniform convergence is then given by:

$$P\left(\sup_{\alpha} |R(\alpha) - R_{emp}(\alpha)| > \varepsilon\right) \leq 4\exp\left\{\left(\frac{G^{\Lambda}}{l} - \frac{\varepsilon^2}{4B}\right) l\right\}, \tag{2.5}$$

where B is an upper bound $< \infty$ on the possible set of functions $Q(z_i, \alpha)$ and G^{Λ} is the so called growth function. This growth function describes the total number of different labellings that can be distinguished by the classifier. The bound on the probability of deviation is distribution independent and a worst case approach [Vap82]. The growth function is of particular interest, since it indicates the capacity of the set of classifiers in Q under investigation. It is closely related to the so-called Vapnik-Chervonenkis (VC) dimension of a classifier. The growth function is bounded as follows:

$$G^{\Lambda}(l) \quad \begin{cases} = & l \ln 2 & \text{if} \quad l \leq h \\ \leq & h(\ln \frac{l}{h} + 1) & \text{if} \quad l > h \end{cases}, \tag{2.6}$$

where h is the VC dimension. The VC dimension is a key issue in the theory of Vapnik and the understanding of generalisation in neural networks. Using this dimension the capacity of a classifier can be determined. Moreover, the VC dimension gives an upper bound on the probability that the real risk and empirical risk do not deviate more than ε (see equation 2.4). This also shows its direct relation to the growth function G^{Λ}.

The VC dimension of a set of classifiers $Q(\vec{x}, \alpha)$ is defined as follows [Vap82]:

> The VC dimension of a set of functions $Q(\vec{x}, \alpha)$ is the maximum number h of vectors $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_h$ that can be separated into two classes in all 2^h possible ways using functions of the set. If for any n (the number of samples) there exists a set of n vectors $\vec{x}$ which can be separated by the set $Q(\vec{x}, \alpha)$, then the VC dimension is equal to infinity.

This dimension is distribution free, it solely depends on the worst data set the classifier can distinguish. However, as shown by Kraaijveld [Kra93], this dimension is usually an overestimate of the capacity of a classifier and in practice the dimensionality is much lower when taking the algorithm to train the classifier and the actual data into account. The determination of the VC dimension of a classifier is not a trivial task. For certain classes of classifiers, e.g. polynomial or linear classifiers, the VC dimension is known but in general neural classifiers do not implement such functions directly or their degree is hard to estimate. Figure 2.2 shows a graphical interpretation of the VC dimension for a particular 2-dimensional classifier with only two degrees of freedom, x and y position. It shows that such a classifier can handle the arbitrary labelling of 4 samples in the 2-dimensional space. In general the VC dimension of linear classifiers is equal to their dimensionality + 1. Devroye [Dev88] discusses the VC dimension for different classifiers which are common in pattern recognition. For instance the VC dimension for the weighted k-NN rule is given by the bound m^k where k is the number of neighbours and m the number of samples. Equation 2.5 containing the growth function reduces now to [Dev88]:

$$P\left(\sup_{\alpha}|R(\alpha)-R_{emp}(\alpha)|\right) = 6VC_Q(2l)\exp\left(-\frac{\varepsilon^2 l}{4}\right), \tag{2.7}$$

where $VC_Q(2l)$ is the VC dimension of the set of classifiers on $2l$ samples. This means that when one has determined the VC of a classifier, its capacity is known. The VC dimension together with the growth function can thus be used to control the speed of the generalisation process.

Vapnik has introduced the support vector classifier (SVC) as a method to try to find the best generalising classifier. The approach assumes that classifiers which have a small capacity will be the best generalising ones on a given finite data set. Section 2.3 will introduce the SVC more formally and discusses its properties.

2.2.3 Compactness

In the previous subsections two views have been presented when dealing with the generalisation property of (neural) classifiers. However, we will adopt another perspective with respect to the distribution of the data and corresponding classifier. In general the data is not completely randomly distributed in the feature space, it has a certain compactness.

Consider figure 2.3, it shows two classes which are separated using a classifier $\mathcal{S}$. The points X_2 and Y_3 are samples which lie close to the class decision borders. If an error in measuring their feature values is made, their class membership may change easily. Such samples are called *border points*, if the classifier is not chosen carefully enough they may be classified

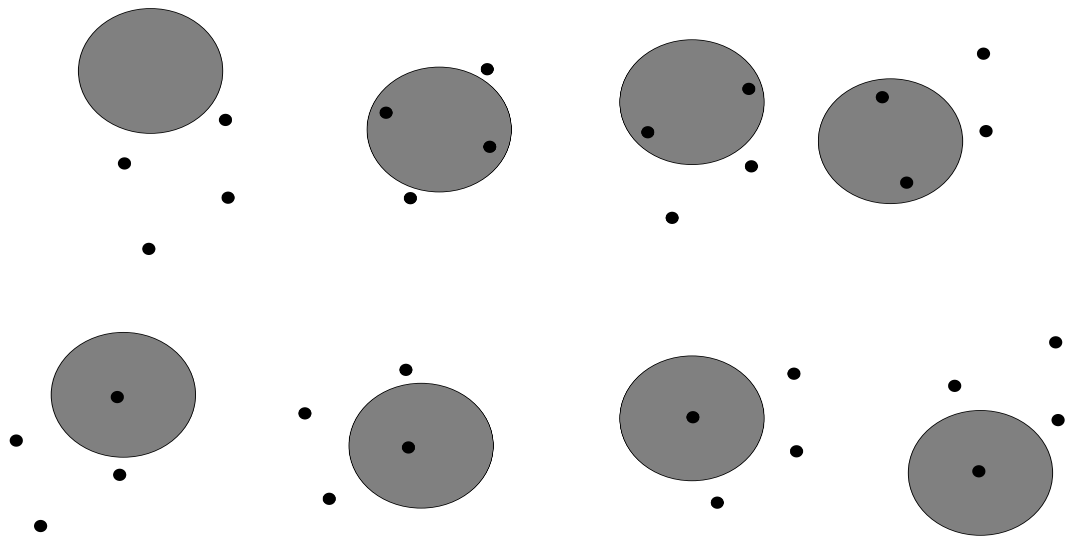


Figure 2.2: *The visualisation of the capacity of a particular classifier in a 2-dimensional space. Only the x and y position are degrees of freedom.*

differently. The points X_1, X_3, Y_1 and Y_2 are not subject to such changes and are therefore referred to as *inner points*. They will (in general) always be classified correctly. Samples within a reasonable neighbourhood of inner points can be connected by a line which will be completely within the class. This constraint is of importance, since in practical situations one is dealing with samples that are derived from noisy measurements. If one does not apply that constraint, a small error in measurement would lead to a sample which is assumed to be of another class. Although some classifiers are more sensitive to this phenomenon than others. The classes in figure 2.3 are said to be *compact* since the number inner points is larger than the number of border points.

This notion of compactness of classification problems was defined by Arkadjew and Brawerman [AB66] in 1966. They used the following definition which we translated from the German issue of their article.

Given a set of samples, when this set of samples fulfils the following three constraints it is said to be *compact*:

1. The number of border samples is small compared to the total number of samples in that class. This implies that relatively few samples lie on the borders between classes compared to the total number of samples in the class.

2. Two randomly chosen inner samples from a class can be connected by a shallow curve that is enclosed by the class. Basically this means that samples that are not on the class border can be connected by almost straight lines.

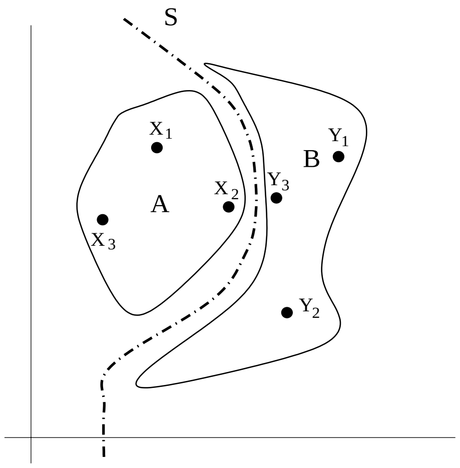


Figure 2.3: *A 2-dimensional data set in which both classes are compact separated using a nonlinear classifier with a low capacity.*

3. Finally, all samples within a reasonably large neighbourhood of an inner sample share the same class label.

A close inspection of this definition shows that the compactness of a class is defined by how it lies in relation to the other classes. If classes share a large boundary their compactness tends to decrease. The dimensionality of the data does not influence the compactness if the class boundaries remain fixed. Figure 2.4 shows two data sets each consisting of two classes $\mathcal{A}$ and $\mathcal{B}$. On the left a data set is shown which is compact as it fulfils all the necessary constraints. If one inspects the label of sample x_1 it will probably be $\omega_{\mathcal{A}}$ due to the compactness. On the right in figure 2.4 a non-compact data set is shown. In Wolpert's framework this is allowed; it is even the case that these situations are more likely to occur. Vapnik does not place a constraint since in the Uniformist view one is interested in the possible labellings that could be assigned and not in the labellings that are given. Note that in Wolpert's framework no constraint is put on the class membership of samples. So these framworks cannot be applied in pattern recognition problems with noisy measurements and therefore one has to follow the compactness approach.

2.3 Generalisation in practice

In the previous section two views on generalisation in (neural) classifiers have been presented. The question which arises is how these viewpoints can be applied to neural network research. Wolpert states that whenever one has a priori knowledge about a problem his theory is not applicable anymore. Zhu and Rohwer [ZR95] showed that if one has no prior knowledge about a problem cross-validation is as good as random guessing. This is counterintuitive since cross-validation is a well established method in situations where one

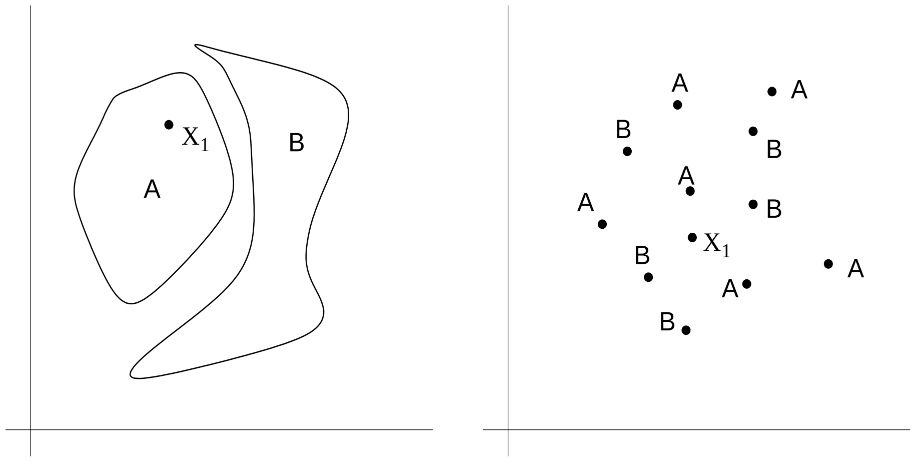


Figure 2.4: *The left situation depicts the compactness of a class, whereas on the right non-compact classes are shown.*

has no prior knowledge. This due to the fact that Wolpert assumes a random labelling of the test set, which has no relation to the learning set. We, however, assume that the test set and learning set are drawn from the same, in general unknown, distribution.

Vapnik's theory is more applicable in practice. There exists a large body of literature on how to compute the VC dimension for neural network classifiers [Dev88, Kra93, Hol96]. A problem, however, is that the VC dimension is usually computed using hard limited neurons. In practice sigmoidal units are chosen. These have far more "soft" decision boundaries and networks consisting of these units are therefore expected to have a larger capacity. Still the bounds derived are based on a worst case approach, since they usually do not take into account the learning algorithm and data to be classified. Kraaijveld showed that for a given algorithm the capacity of a neural network classifier is far less than the value expected from theory. He introduced the effective capacity VC_{eff} and actual capacity VC_{act}. The effective capacity is used to take into account the capacity reduction induced by the choice of a certain learning algorithm. When taking also the data set into account, one can estimate the actual capacity. This capacity reduction is to be expected. The choice of the learning algorithm will restrict the number of possible classifiers to be considered. Learning algorithms also depend on the initialisation and the architecture of the network. Taking into account the data leads to a reduction since it is unlikely that it is the "worst case" data set.

Consider figure 2.3. Here two classes are given that can be perfectly separated. The capacity needed to arbitrarily label this data set will be huge. However, if it is possible to find a low capacity classifier to separate the classes it is expected to generalise. Finding such a classifier is a difficult problem. When inspecting the two classes in more detail, only the touching part of the classes is of interest, the border samples. This notion of only a limited number

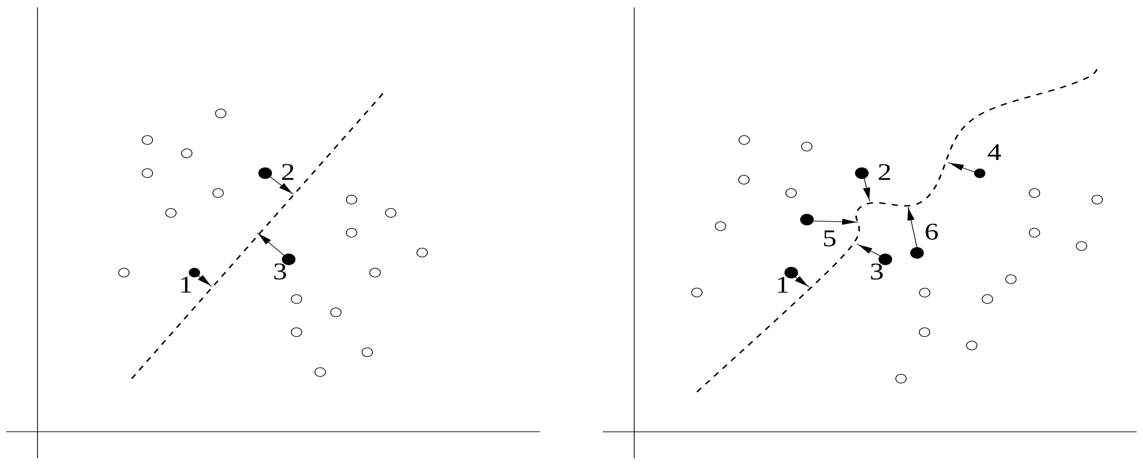


Figure 2.5: *Two identical data sets separated using two different classifiers. The numbered samples are those samples which are support for the classifier.*

of samples influencing the classifier led to the introduction of the Support Vector Classifier (SVC) by Vapnik [Vap95]. This classifier is based on a small subset of the data set. They are called support vectors since they alone determine the description of the classifier. In figure 2.5 this is graphically illustrated. A linear classifier only needs three samples as support, these samples are enough to describe the whole classifier. The other classifier needs much more support since it is nonlinear. This basically illustrates what a support vector classifier is, it is described by those samples from a training set which offer enough support to induce a decision function. Now the support classifier will be introduced more formally. A complete description of the SVC and its applications can be found in Vapnik [Vap95] and Schölkopf [Sch97] on which the following descriptions are based.

The basis of the SVC is to find the best separating hyperplane which separates the set of samples with the lowest error. In other words, the SVC tries to find the best generalising classifier, i.e. the best non-overtrained classifier (see section 1.2). This hyperplane can be calculated in an arbitrary space. That is, one is not restricted to determining the hyperplane in the feature space. By a transformation, a larger space can be created in which a linear separation is possible. The training set is denoted in the following way:

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_l, y_l) \tag{2.8}$$

where $\vec{x}_i$ denotes a sample to be classified or trained with and $y_i \in \{-1, 1\}$ its corresponding label. The linear hyperplane which separates the samples is given by [Vap95]:

$$y_i[(\vec{w}\vec{x}_i^T) + b] \quad \geq \quad 1, \quad i = 1, \ldots, l. \tag{2.9}$$

The training set is said to be linearly separable if there exists a $\vec{w}$ such that equation 2.9 holds for all samples in the training set. The optimal hyperplane is that plane which separates the training data according to equation 2.9 and for which the distance to the nearest training sample is maximal. Those vectors which result in $y_i[(\vec{w}\vec{x}_i^T) + b] = 1$ are the support vectors. This hyperplane can be found by minimising the following functional, under the condition given by equation 2.9:

$$\Phi(\vec{w}) = \frac{1}{2}(\vec{w}\vec{w}^T). \tag{2.10}$$

This results in the following description of the optimal hyperplane:

$$\vec{w}_{best} = \sum_{i=1}^{l} y_i \alpha_i^{best} \vec{x}_i, \tag{2.11}$$

where $\alpha_i^{best} \geq 0$. Only those samples which are support vectors will have an $\alpha_i > 0$, any other non-support vector will have $\alpha_i = 0$. The aim now is to find the appropriate vector $\vec{\Lambda}_0 = (\alpha_1^0, \alpha_2^0, \ldots, \alpha_l^0)$ which gives the best separating hyperplane. This can be done by solving [CV95]:

$$W(\vec{\Lambda}) = \vec{\Lambda}^T I - \frac{1}{2}\vec{\Lambda}^T \mathbf{D} \vec{\Lambda}, \tag{2.12}$$

where $\mathbf{D}$ is a symmetric $l \times l$ matrix with elements:

$$\mathbf{D}_{ij} = y_i y_j \vec{x}_i \vec{x}_j^T, \quad i, j = 1, 2, \ldots, l, \tag{2.13}$$

and subject to the constraints:

$$\vec{\Lambda} \geq \vec{0}, \tag{2.14}$$

$$\vec{\Lambda}^T \vec{Y} = 0, \tag{2.15}$$

where $\vec{Y} = (y_1, y_2, \ldots, y_l)$. The solution of equation 2.12 can be obtained by using standard minimisation methods. Tax and De Ridder [TdRD97] have implemented a SVC in Matlab using the optimisation toolbox and PRTools [Dui95a]. Experiments shown later in this thesis were obtained by using this implementation. Note, however that the resulting optimal hyperplane can be used on perfectly separable classes only. Equation 2.9 has to be changed in case of overlapping classes. Overlapping classes will inherently cause classification errors. By modifying the equation to:

$$y_i[(\vec{w}\vec{x}_i^T) + b] \quad \geq \quad 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, 2, \ldots l. \tag{2.16}$$

it allows for a certain error in classification. This classifier is also referred to as the *soft margin* classifier. This means that one allows a certain margin for wrong classification. Equation 2.10 changes to:

$$\Phi(\xi) \quad = \quad \sum_{i=1}^{l} \xi_i^{\sigma}, \tag{2.17}$$

and consequently the minimisation problem to:

$$W(\vec{\Lambda}, \delta) \quad = \quad \vec{\Lambda}^T \mathbf{I} - \frac{1}{2}\left[\vec{\Lambda}^T \mathbf{D} \vec{\Lambda} + \frac{\delta^2}{C}\right], \tag{2.18}$$

where **I** is the unity matrix, subject to the following constraints:

$$\vec{\Lambda}^T \vec{Y} \quad = \quad 0, \tag{2.19}$$
$$\delta \quad \geq \quad 0, \tag{2.20}$$
$$0 \quad \leq \vec{\Lambda} \leq \quad \delta \mathbf{I}, \tag{2.21}$$

and C sufficiently large. The constant C determines how severe an error must be accounted for.

The methods described result in a hyperplane that is located in the feature space. However, it can be the case that the training data is not linearly separable in that space. Therefore Vapnik proposes a method to transform the feature space to a higher dimensional space in which a linear separation is possible. The computation of the SVC in a transformed space equals the computation of a nonlinear separation function in the feature space. For example a 2-dimensional space, described by features z_1 and z_2, can be transformed to a quadratic space. This is done by introducing the "new" features z_1^2, z_2^2 and $z_1 z_2$. More formally, assume a transformation function function $\phi : \mathcal{R}^n \to \mathcal{R}^N$. Consequently all learning samples can be transformed to:

$$\phi(\vec{x}_i) \quad = \quad \phi_1(\vec{x}_i), \phi_2(\vec{x}_i), \ldots, \phi_N(\vec{x}_i), \quad i = 1, 2, \ldots, l. \tag{2.22}$$

The classification function also changes, it uses the transformation function to classify:

$$f(\vec{x}) \;=\; \vec{w} \cdot \phi(\vec{x}) + b \tag{2.23}$$

The $\vec{w}$ can again be solved by using the support vector approach by writing $f(\vec{x})$ as:

$$f(\vec{x}) \;=\; \sum_{i=1}^{l} y_i \alpha_i \phi(\vec{x}_i) \cdot \phi(\vec{x}_i) + b. \tag{2.24}$$

The α's can be solved using the equations above. However, the dot product $\phi(\vec{x}_i) \cdot \phi(\vec{x}_i)$ has to be valid. This can be guaranteed by using a generalised dot product $\phi(u) \cdot \phi(v) \equiv K(u, v)$. Here K is a kernel function that needs to satisfy Mercer's theorem, i.e. it has to be positive on its domain and its L_2 norm has to be bounded [CV95]. As long as one uses such a kernel almost any classification problem can be solved using a SVC, provided one uses the appropriate degree of the kernel function. This leads to separation functions that have a low VC dimension and are therefore expected to generalise very well. Experiments [Vap95, Sch97] show that this is indeed the case.

By transformation of the original space into another one using a kernel function, a best linear hyperplane in that space can be found using the techniques described above. It also enables us to perform featureless classification [DdRT97, DdR97], by taking a kernel function that computes a similarity measure between objects in some feature space. Vapnik also points out that the SVC can be used for regression by taking splines as kernel functions. The SVC is a very powerful classifier and it will be used for referential purposes in the coming chapters.

2.4 Discussion

In this chapter two viewpoints about generalising classifiers were presented. On one hand there is the theory of Wolpert which takes the extended Bayesian framework as basis. In this framework one is only interested in the probability of data. Classifiers are uniformly distributed. However, if one can state something a priori about the data and classifier then Wolpert's theorems do not hold.

On the other hand there is Vapnik's approach which takes the classifier as starting point. Here we are not interested in the distribution of the data, but only in the number of samples we have. Depending on the classifier, an estimate of its capacity can be made, i.e. the number of dichotomies it can induce on the samples. This capacity is distribution free and even in its original formulation learning algorithm independent. Kraaijveld showed that when taking into account the learning algorithm and the actual data distribution the capacity is much lower.

We adopted the compactness approach since we have knowledge about the physical processes that generates the data. This implies that a priori assumptions about the distribution of the data can be made. Moreover, the data sets obtained are not the worst case ones as expected by Vapnik. This results in the need for classifiers having a relatively low complexity and a high generalisation capability. The problem how to find such a classifier still remains. One way is to use the SVC which results in a classifier having a large generalisation capability. The remaining chapters in this thesis will present techniques with which a classifier can be found that is an appropriate one in terms of generalisation with respect to the data that it has to classify.

Chapter 3

Classifier Nonlinearity

3.1 Introduction

The previous chapter introduced a theoretical study of generalisation in (neural) classifiers. This chapter a measure for the nonlinearity of a classifier is presented that can be used to study training behaviour. In many experiments it was suspected that classifiers start from a linear solution and gradually become nonlinear. As they grow more nonlinear they tend to adapt to the noise in the data set which decreases the generalisation capability. This is an important topic since neural classifiers can implement almost any function [Fun89, HSW89]. However, overtrained classifiers (see section 1.2) are expected not to generalise very well [BH89, VKA94, Wol94], which makes such a classifier useless for solving the classification problem, see for instance [Dui93b, KK82] and chapter 2. Parts of the research presented in this chapter have been taken from [HD96].

In this chapter the shape of the classification function implemented by the network is related to the (over)training behaviour. This will be referred to as nonlinearity of the classifier. It is closely related to the concept of compactness as introduced in section 2.2.3 since border points influence the nonlinearity. Furthermore, it will be shown in this chapter that overtrained classifiers show a larger nonlinearity than optimal ones. Moreover, the nonlinearity is also applicable to traditional classifiers, e.g. k nearest neighbour classifier, which enables us to compare neural networks and traditional classifiers on given data. The nonlinearity measure presented in this chapter is related to a known complexity measure like the actual capacity [KD94] (see chapter 2), which relates the capacity of a classifier to the training procedure and the data it is trained with.

This chapter starts with a section in which the nonlinearity measure will be introduced both in an informal and a formal way. It also describes the algorithm used to compute the nonlinearity. Section 3.4 discusses the connection between the support vector classifier and the

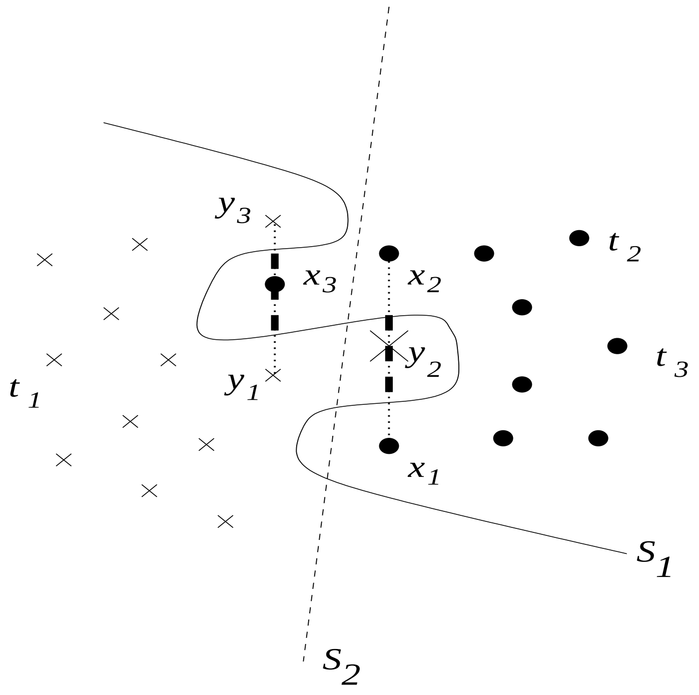


Figure 3.1: *A highly nonlinear and a linear decision function on a set of samples*

nonlinearity measure, followed by a section which shows the experimental results obtained by applying the nonlinearity measure to test problems. The chapter ends with a discussion.

3.2 A nonlinearity measure

3.2.1 Informal model

As stated earlier, the classifier adapts itself to the data, therefore we relate the nonlinearity of a classifier with respect to the data it has to classify. Consequently nonlinearities that are not observed in the classification result are not interesting. This can be the case for the behaviour of the classifier in areas where no data is present or on a small scale between samples, but not influencing their classification.

Figure 3.1, derived from [HD96], shows a set of samples and two classifiers $\mathcal{S}_1$ and $\mathcal{S}_2$. The behaviour of both classifiers is the same for samples in a remote area of the data space, i.e. far from the classifier's decision boundary. In that case both classifiers are interchangeable, samples are classified the same for both $\mathcal{S}_1$ and $\mathcal{S}_2$ (as the samples t_1 and t_2 in figure 3.1). Moreover, for remote samples it holds that any linear combination of two samples, i.e. of their feature vectors, sharing the same classification results in the same classification. That is, all interpolated samples have the same classification. In figure 3.1 this can be observed when interpolating between samples t_2 and t_3. Any of the interpolated points will be assigned the same class as t_1 and t_3. Moreover, the classifier is of no influence. These kind of points are *inner points* as mentioned in chapter 2, section 2.2.3. This situation changes for samples located near to the classifier boundary.

Consider again figure 3.1. Samples that lie between y_1 and y_2 have the same classification with respect to classifier $\mathcal{S}_1$. However, samples that lie on a linear interpolation between y_1 and y_3 (or x_1 and x_2), depicted by the dotted lines, do not share the same classification. Here a fraction of the interpolated samples is assigned to a different class than that of y_1 (or y_3), depicted by the fat dashed line in figure 3.1. In other words, here a nonlinearity of the classifier $\mathcal{S}_1$ is observed. If one would have chosen $\mathcal{S}_2$ instead of $\mathcal{S}_1$, none of the nonlinearities observed with $\mathcal{S}_1$ would occur. Note that the relation between classifier and the data is of particular importance. Suppose the samples $x_1, \ldots, x_3$ and $y_1, \ldots, y_3$ are removed, then the nonlinearity of $\mathcal{S}_1$ will not be observed since all samples and their linear interpolations share the same classification.

These observations lead to the following definition of nonlinearity:

Definition 2 (*Nonlinearity* **)** :
The nonlinearity $\mathcal{N}$ of a classifier $\mathcal{S}$ with respect to a data set $\mathcal{L}$ is the probability that an arbitrary point, uniformly and linearly interpolated between two arbitrary samples in the data set with the same classification does not share this classification. □

This definition of the nonlinearity is independent of the true class labels of the samples since only the classification made by the classifier is taken into account. Moreover, due to the independence of the true class labels, the nonlinearity is also independent of the classifier performance. However, it does depend on the dataset used to train the classifier. Now the nonlinearity of a classifier is introduced more formally.

3.2.2 Formal model

First we define the nonlinearity of a classifier $\mathcal{S}$ with respect to two samples x_k and x_l sharing the same classification, e.g. x_1 and x_2 in figure 3.1. In the following definition the nonlinearity contribution for two samples, $n((x_k, x_l))$ (assuming $x_k \neq x_l$) is stated.

Definition 3 (*Nonlinearity contribution of two samples***)** :

$$\begin{aligned} n((x_k, x_l)) &= P(\mathcal{S}(\alpha x_k + (1-\alpha)x_l) = \mathcal{S}(x_k)), \quad \alpha \approx U(0,1) \\ &= \frac{1}{||x_k - x_l||} \int_0^1 \mathcal{S}_\Delta(\alpha x_k + (1-\alpha)x_l, x_k) d\alpha, \end{aligned}$$

□

where (x_k, x_l) is a pair of samples having the same classification. The term $1/||x_k - x_l||$ is used to normalise for the distance between two samples. Between the samples x_k and x_l a linear interpolation is done by $\alpha x_k + (1-\alpha)x_l$. Each of the samples in the interpolation is checked against the class of x_k using $\mathcal{S}_\Delta$:

$$\mathcal{S}_\Delta(a,b) \;=\; \begin{cases} 0 & \text{if} \quad \mathcal{S}(a) = \mathcal{S}(b) \\ 1 & \text{otherwise} \end{cases} . \tag{3.1}$$

Consequently, if all interpolated samples share the same classification, no nonlinearity is observed and therefore the nonlinearity will be zero. Any change in classification increases the nonlinearity, the size of the increment is determined by the fraction of differently classified samples.

Now the nonlinearity of $\mathcal{S}$ with respect to a data set $\mathcal{L}$ can be defined using the definition of $n((x_k, x_l))$.

Definition 4 (*Nonlinearity of a classifier with respect to a data set***)** :

$$\mathcal{N}(\mathcal{L},\mathcal{S}) \;=\; \frac{1}{\sum_{i=1}^{M} |\mathcal{L}^i \times \mathcal{L}^i| - |\mathcal{L}^i|} \sum_{i=1}^{M} \sum_{\substack{(x_k, x_l) \in \mathcal{L}^i \times \mathcal{L}^i \\ x_k \neq x_l}} n((x_k, x_l)),$$

□

where $\mathcal{N}(\mathcal{L},\mathcal{S})$ is the nonlinearity of a classifier with respect to a data set, $\mathcal{L}^i$ is the set of samples sharing the same classification and $\mathcal{L}^i \times \mathcal{L}^i$ the Cartesian product of the sets. The M denotes the number of classes present. Since no combinations of identical samples is allowed, the number of available pairs in $\mathcal{L}^i \times \mathcal{L}^i$ is not $|\mathcal{L}^i \times \mathcal{L}^i|$ but $|\mathcal{L}^i \times \mathcal{L}^i| - |\mathcal{L}^i|$. The division of $\mathcal{L}$ in $\mathcal{L}^i$s is needed since we interpolate between samples of the same assigned class label. The nonlinearity $\mathcal{N}(\mathcal{L},\mathcal{S})$ is calculated by interpolation between all possible pairs of samples in the data set, provided that they have the same assigned class labels.

The integral expression in definition 3 can be approximated using a sum expression, it can therefore be rewritten as:

$$n((x_k, x_l)) \;=\; \frac{1}{||x_k - x_l||} \sum_{j=0}^{N} S_\Delta \left(\left[\frac{j}{N} \right] x_k + \left(1 - \left[\frac{j}{N} \right] \right) x_l, x_k \right). \tag{3.2}$$

The sum will approximate the integral expression for large N. In practical situations N can be chosen such that a required accuracy for the computation can be achieved, e.g. a typical value for N might be 10,000. The equation in definition 4 can be rewritten to:

$$\mathcal{N}(\mathcal{L},\mathcal{S}) = \frac{1}{M}\sum_{i=1}^{M} \cdots \frac{1}{|\mathcal{L}' \times \mathcal{L}'| - |\mathcal{L}'|} \sum_{\substack{(x_k, x_l) \in \mathcal{L}^i \times \mathcal{L}^i \\ x_k \neq x_l}} \cdots \frac{1}{||x_k - x_l||} \sum_{j=0}^{N} S_\Delta\left(\left[\frac{j}{N}\right] x_k + \left(1 - \left[\frac{j}{N}\right]\right) x_l, x_k\right). \tag{3.3}$$

Note that the term $1/\sum_{i=1}^{M} |\mathcal{L}^i \times \mathcal{L}^i| - |\mathcal{L}^i|$ in definition 4 has been reformulated in the equation to $1/M \sum 1/|\mathcal{L}' \times \mathcal{L}'| - |\mathcal{L}'|$ under the assumption $|\mathcal{L}'| = |\mathcal{L}^i|, \forall i$. The integral in definition 3 has been replaced by a approximation using a summation. Equation 3.3 can be easily implemented in a computer environment. However, computing the $\mathcal{N}(\mathcal{L},\mathcal{S})$ may take a huge amount of computation time. Assuming an equal number of samples per class, i.e. $L = |\mathcal{L}^1| = |\mathcal{L}^2| = \ldots = |\mathcal{L}^M|$ and a restriction to two class problem results in the following possible definition of a computational complexity cost function:

$$\text{Computational costs} = (L^2 - L) \cdot (N + 1) \cdot M + C, \tag{3.4}$$

where C is a constant which compensates for the normalisation (the $1/\cdots$ terms). Typical numbers for a computation would be, $M = 2$, $N = 10,000$ and $L = 1,000$. The resulting computational costs would be $20 \cdot 10^9$. This gets even worse when calculating the nonlinearity at different stages of the training phase of the classifier. Therefore, we constructed an algorithm that uses a Monte-Carlo technique to estimate $\mathcal{N}(\mathcal{L},\mathcal{S})$. Below the algorithm is stated:

1. Classify all samples according to the classifier $\mathcal{S}$.
2. Randomly draw N samples x_k from the data set $\mathcal{L}$.
3. Find for each x_k, a corresponding x_l having the same classification label, i.e. create a pair (x_k, x_l).
4. Randomly generate α's (where $0 \leq \alpha \leq 1$) for each of the N pairs (x_k, x_l).
5. Compare the classification $\mathcal{S}(\alpha x_k + (1 - \alpha)x_l)$ with $\mathcal{S}(x_k)$, and count as nonlinearity if they are not classified in the same class. Do this for all pairs (x_k, x_l).
6. $\mathcal{N}(\mathcal{L},\mathcal{S})$ is now determined by the number of nonlinearity counts divided by N.

The algorithm approximates $\mathcal{N}$ by taking N trials. In the experiments described in section 3.3, the above algorithm was used.

3.3 Experiments

In order to observe the nonlinear behaviour of classifiers and to relate it to their generalisation behaviour, several experiments were conducted. One of the experiments shows the nonlinear behaviour of a traditional classifier. This classifier yields a stable nonlinearity, i.e. it needs to be calculated just once, and can therefore serve as a reference for the neural classifier in the experiment since a neural classifier yields a non-stable nonlinearity. The data sets used were 2-dimensional in order to be able to visually inspect the behaviour. The nonlinearity measure, however, is not restricted to these cases. In figure 3.2 the two data sets are shown which were used throughout the experiments.

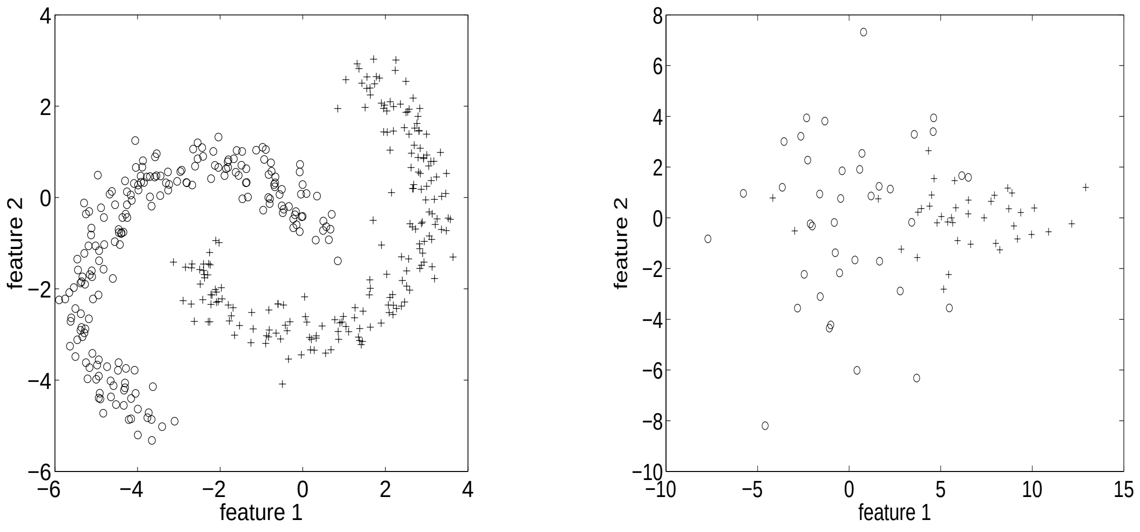


Figure 3.2: *The data sets used in the nonlinearity experiments, one having no overlap ('Banana') on the left and one with an overlap ('Overlap') on the right.*

The data set on the left (figure 3.2) shows a perfectly separable data set but it would require a highly nonlinear function to do it. On the right (figure 3.2) a strongly overlapping data set is depicted. However, it requires only a quadratic function for an optimal separation given the distribution of the samples. Both classes are Gaussian distributed, with one class ($\circ$) having mean $(0,0)$ and the identity matrix as covariance matrix. The other class ($+$) has mean $(6,0)$ and a variance of 4 for both features. The sets also differ in the number of samples used for training. This was done to study the effect of the training algorithm on the nonlinearity.

Throughout this section the data sets will be referred to as 'Banana' and 'Overlap'.

On the 'Banana' set a k nearest neighbour classifier and a neural network were trained. The k voor the nearest neighbour ranged from 1 to 200, the neural network was trained with the Levenberg-Marquardt learning rule and had 2, 4, 10 and 20 hidden units. In the experiment with the 'Overlap' data set, a neural network and a quadratic classifier were used. The neural network was trained using again the Levenber-Marquardt rule and had 2,4,8,10,20 and 30 hidden units.

3.3.1 Banana data set

The 'Banana' set was used to train a k nearest neighbour classifier. As stated, this data set requires a highly nonlinear decision function, but is perfectly separable. First the optimal k and its corresponding $\mathcal{N}$ was calculated. Second, the $\mathcal{N}$ for different values of k was determined. The best k proved to be 1, which is not surprising for separable classes. Figure 3.3 shows the *1* nearest neighbour classifier on the 'Banana' set. Its nonlinear nature is clearly shown. The corresponding $\mathcal{N}$ was calculated and turned out to be 0.11.

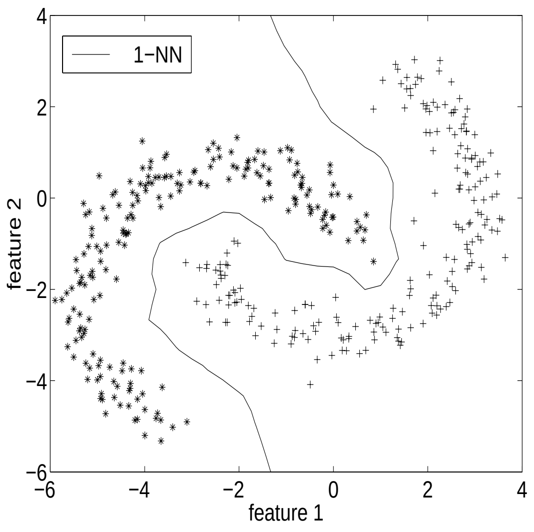


Figure 3.3: *The decision function implemented by the 1 nearest neighbour classifier which is the best one for the 'Banana' set.*

As shown in figure 3.4 the nonlinearity of the nearest neighbour classifier decreases when k becomes larger. This implies that the classifier becomes more linear. This is indeed the case as can be seen in figure 3.5. This figure shows the nearest neighbour decision function for larger values of k. As k increases the decision function implemented becomes "straighter".

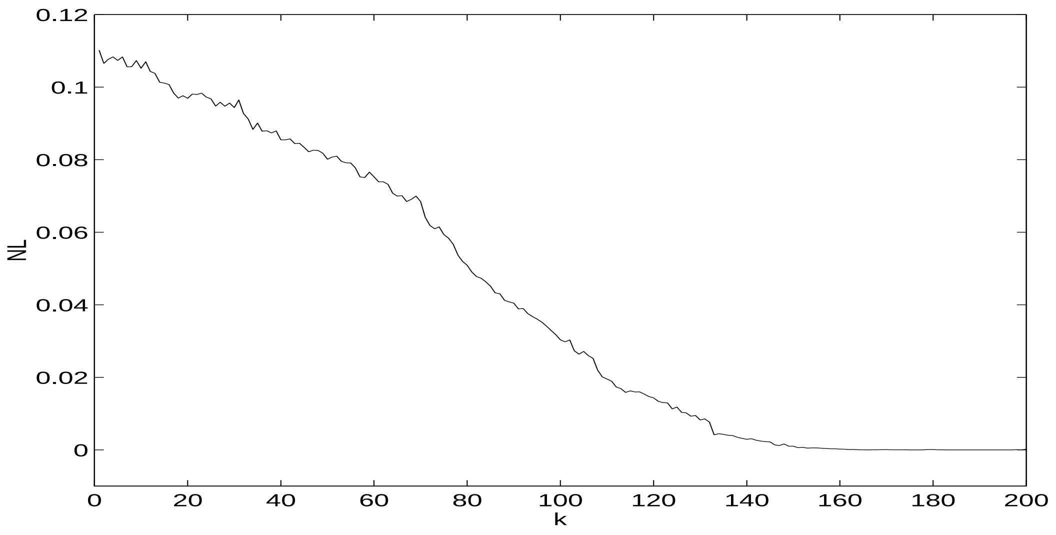


Figure 3.4: *The nonlinearity behaviour for the k nearest neighbour classifier, for different k.*

A large *k* implies that many neighbours are taken into account for classification and therefore it is less likely that nonlinearities are observed.

A set of neural networks with varying numbers of hidden units was trained on this data set. It was expected that, since the classes are perfectly separable, a neural network consisting of enough hidden units would not suffer from overtraining. The network was trained using the Levenberg-Marquardt rule and consisted either of 2, 4, 10 or 20 hidden units. Figure 3.6 shows the decision functions for 2 and 10 hidden units. Clearly two hidden units is too small, resulting in a weakly nonlinear classifier (which is confirmed by the nonlinearity curve in figure 3.7). The 10 hidden units network implements almost the same decision function as the *1* nearest neighbour classifier and has, therefore, an almost identical nonlinearity value of 0.096, where the nearest neighbour classifier had 0.11.

Figures 3.7 and 3.8 show the development of the nonlinearity of the networks during training. Each network was trained for a fixed number of epochs, namely 150. The figures show the results per 5 epochs. Remarkably the nonlinearity of the larger neural network starts high (figure 3.7 on the right) and then stabilises to a lower nonlinearity value. Probably a better network is already found in an early stage of the training. Inspection of the decision function at that stage indeed shows that this function gives a better separation. Figure 3.7 (shown on the right) also shows two decision functions. The solid line shows the decision function found after a first update step. It is far more nonlinear than the dashed function which is the result after several steps. The dashed function does not change much during the training and consequently the nonlinearity remains more or less constant after the “drop”. For the decision function of the 2-hidden unit network (figure 3.8) the opposite holds. The

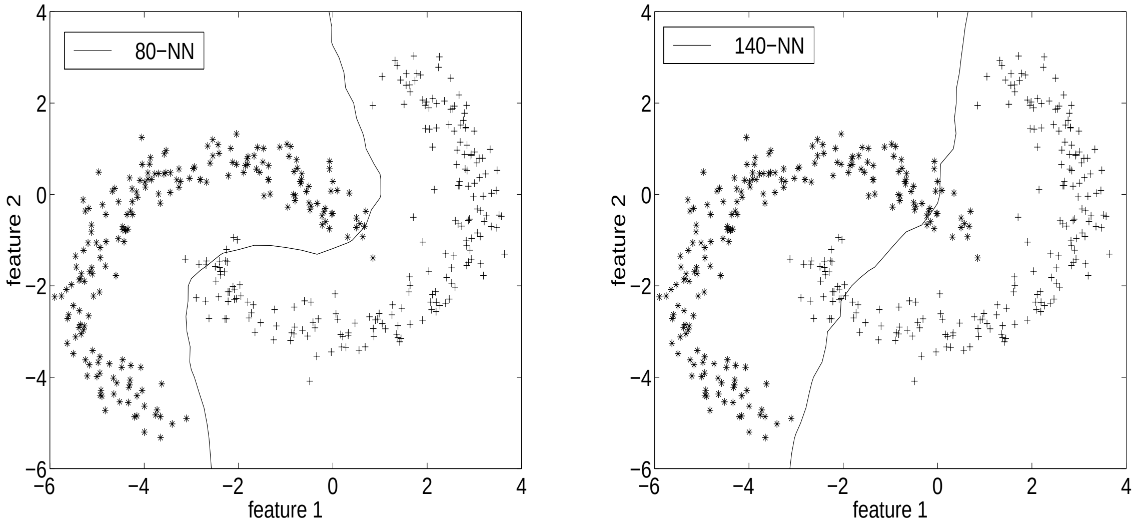


Figure 3.5: *The decision function implemented by the 80 nearest neighbour classifier (on the left) and the function implemented by the 140 nearest neighbour classifier (on the right).*

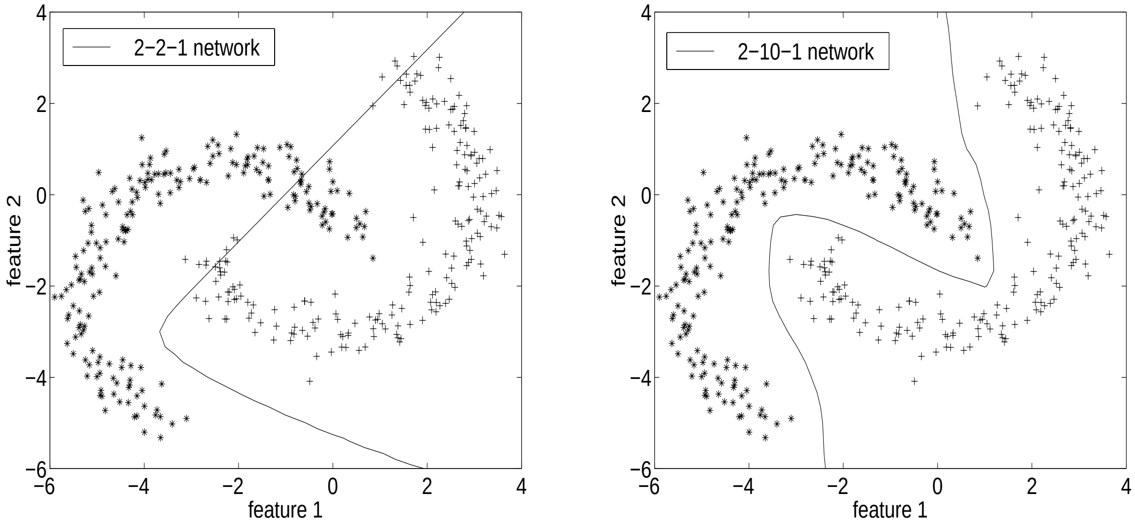


Figure 3.6: *The decision function implemented by the network with 2 hidden units (on the left) and the function implemented by the 10 hidden units network (on the right).*

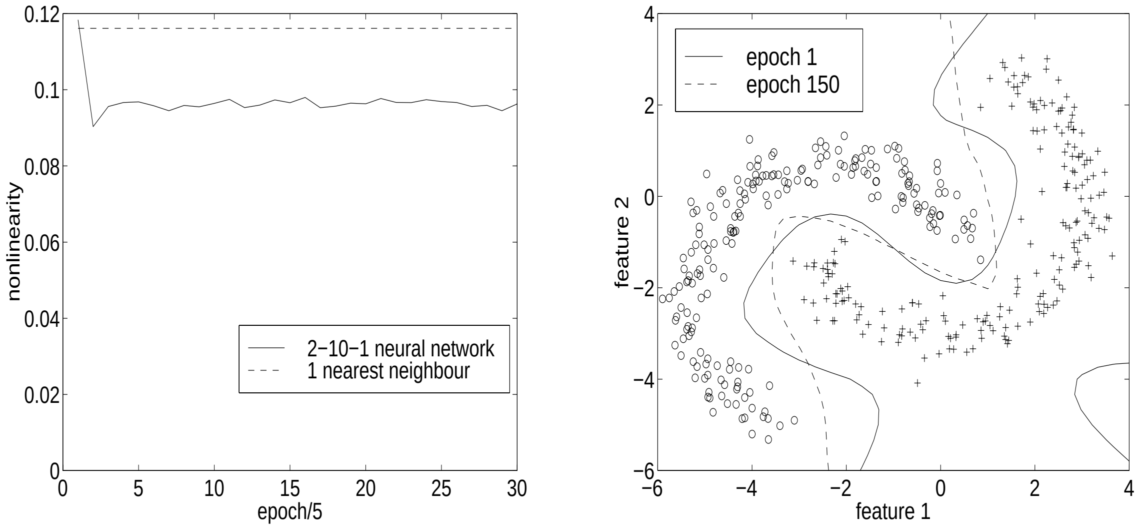


Figure 3.7: *The nonlinearity development of the neural network with 10 hidden units and the 1 nearest neighbour classifier on the left. On the right the decision functions implemented by the neural network are depicted for epoch 1 and 150.*

nonlinearity starts low and quickly rises and stabilises. Due to the small capacity of the 2-hidden unit network no satisfactory solution can be obtained and the resulting $\mathcal{N}$ remains low. This also indicates that the nonlinearity of the nearest neighbour classifier may be used as reference.

3.3.2 Overlap data set

Like almost any other classifier, neural classifiers suffer from small sample size problems. Building the classifier using too few samples will generally result in classifiers having a poor generalisation capability. They have adapted too much to the noise in the data set. It is therefore expected that they have a higher nonlinearity, since the decision function implemented by the classifier is highly nonlinear.

The 'Overlap' set is a small training set of 40 samples per class. It was used in order to investigate the nonlinear behaviour of an overtrained neural network. (Over)training in the neural network was obtained by using the Levenberg-Marquardt (LM) learning rule. This learning rule combined with a sufficient number of hidden units and a small number of examples leads relatively quickly to an overtrained network. This overtraining is intentionally forced since the nonlinearity of the classifier during the whole learning process is of interest and not only that of the optimal network. However, the number of hidden units is difficult

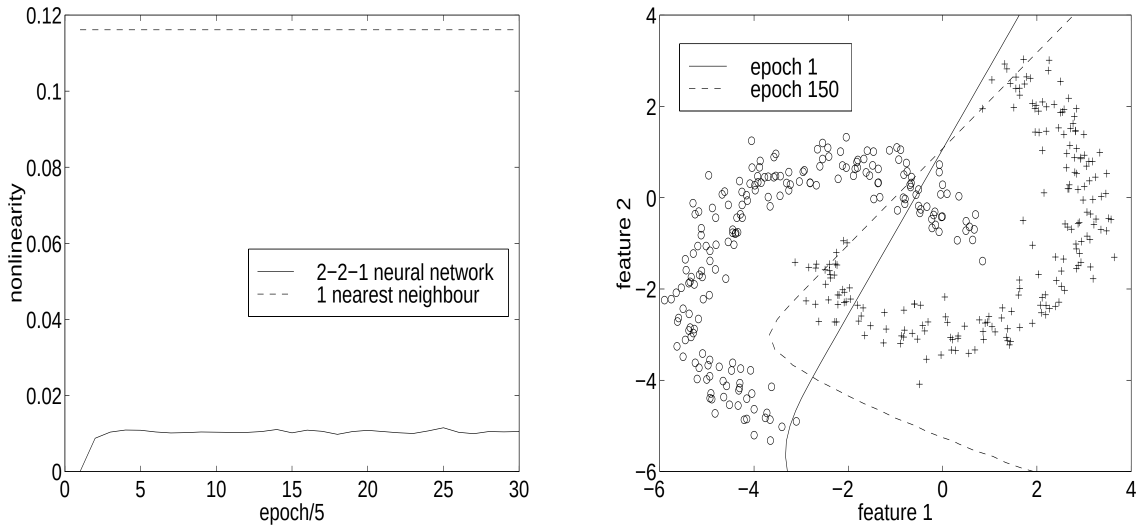


Figure 3.8: *The nonlinearity development of the neural network with 2 hidden units and the 1 nearest neighbour classifier on the left. On the right the decision functions implemented by the neural network are depicted for epoch 1 and 150.*

to choose beforehand. Therefore, several neural networks with varying sizes of the hidden layer were trained and investigated. Six sizes of the hidden layer were inspected: 2,4,8,10,20 and 30. Each network was trained for a fixed number of epochs which was set to 150. For each of these neural networks the nonlinearity was computed on 10,000 data pairs. That is 10,000 pairs of points were randomly selected from the learning set (N=10,000 when using the algorithm shown on page 40). Also a quadratic decision function was calculated on the data set. The nonlinearity of this (optimal) decision function, given the distributions of the classes, was used to serve as a reference, since its nonlinearity is fixed ($\mathcal{N} = 0.004$). It is calculated only once, whereas the nonlinearity of the neural network is calculated during the training process.

Figure 3.9 depicts the nonlinearities of the different networks. Each network is denoted by 2-X-1 indicating that it has 2 inputs, X hidden units and 1 output unit. The nonlinearity for each network starts small and then almost monotonically increases to a certain level where it remains more or less stable. However, this does not occur in all situations. For instance, for the 2-8-1 network, the nonlinearity increases and does not reach a stable point. When the curve of the 2-20-1 network is compared with the error curves (see figure 3.11), it turns out that the nonlinearity increases with increasing generalisation error. Further, when the test set error remains almost constant the nonlinearity also remains stable. Hence the nonlinearity is an indicator for the network's generalisation situation.

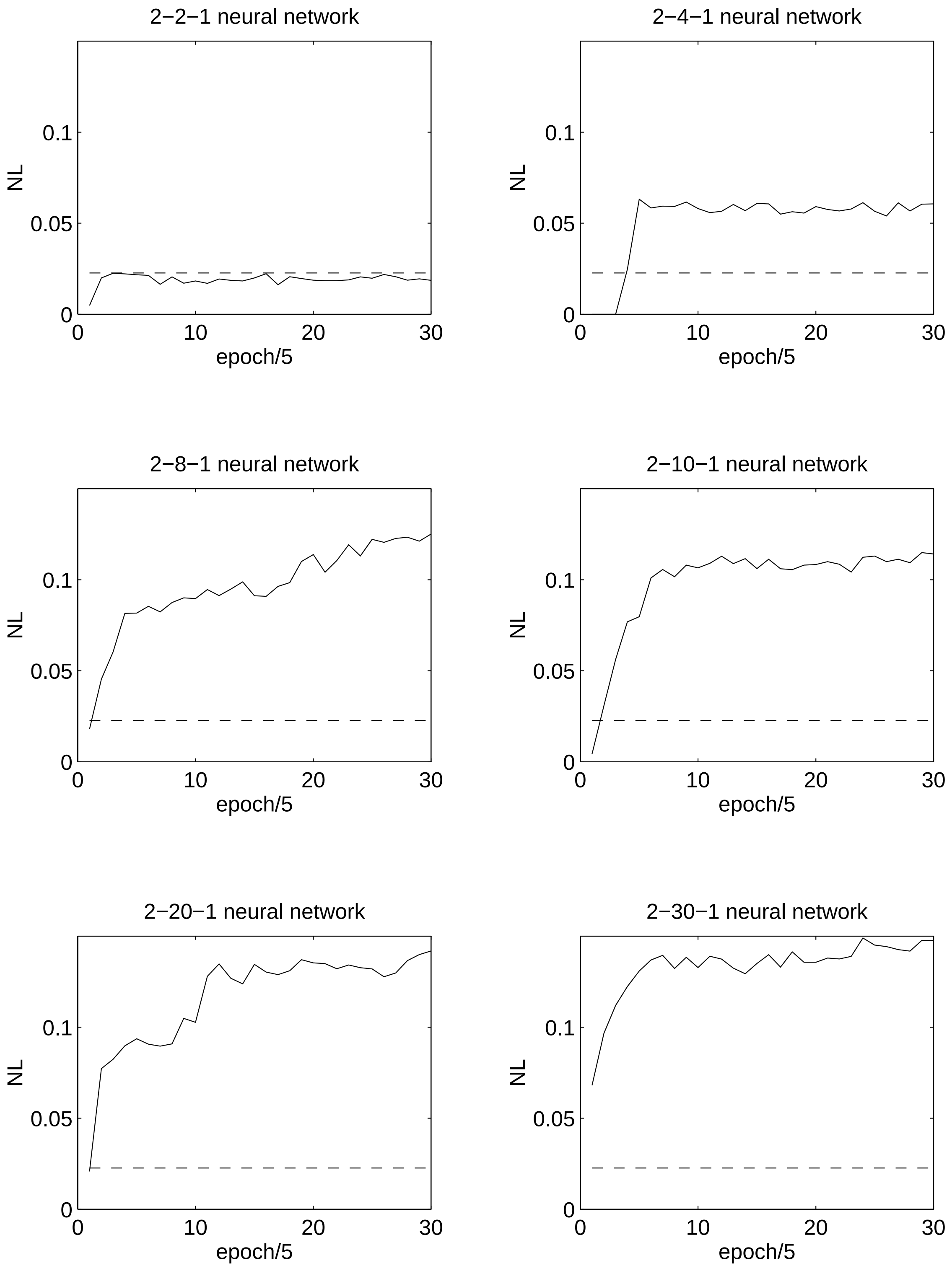


Figure 3.9: *The different nonlinearity curves for networks with various number of hidden units. The dashed line is the nonlinearity of the quadratic classifier ($\mathcal{N} = 0.004$), which is stable for each epoch.*

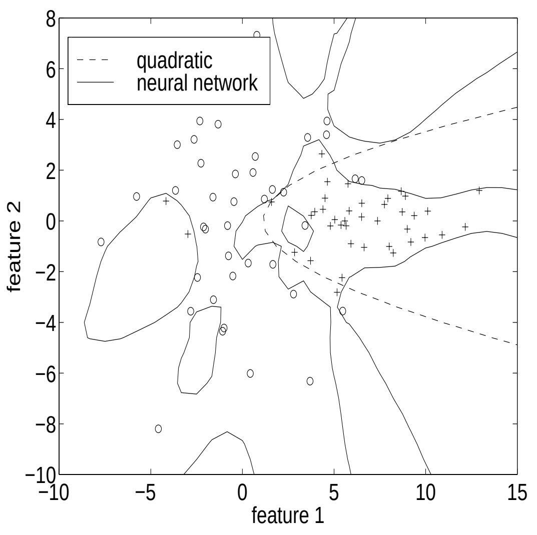


Figure 3.10: *An overtrained network on the overlapping classes. The dashed line denotes the best, quadratic decision function.*

Figure 3.10 depicts the 2-20-1 network's decision function and the optimal decision function using a quadratic classifier. This network is clearly in a severely overtrained situation which implies a high nonlinearity. Compared to the "optimal" nonlinearity it is about 6 times higher. The network with 20 hidden units has far too much flexibility. For instance the 2-2-1 neural network has just enough flexibility since its nonlinearity is almost equal to that of the quadratic classifier.

In figure 3.11 the error curves for the 2-20-1 neural network are displayed. It is clear that the network reaches an overtrained situation due to the increasing test set error and zero training set error. This behaviour of the generalisation curve is reflected in the nonlinearity curve. Up till epoch 50, $\mathcal{N}$ gradually increases just as the test set error, at about epoch 50 there is a large increase in nonlinearity and a relatively large increase in the test set error. The training set error behaves just the other way around, at epoch 50 it drops to a lower level resulting in a test set error increase.

3.4 VC dimension and nonlinearity

In the previous chapter the VC dimension for classifiers was introduced. It indicates the capacity of a classifier with respect to the number of samples to be classified. The larger its capacity, the less it is able to generalise. By taking into account the effective capacity and the actual capacity the VC dimension can be related to a particular classifier and data set. In cases where one has a high VC dimension, the classifier is likely to adapt too much to the

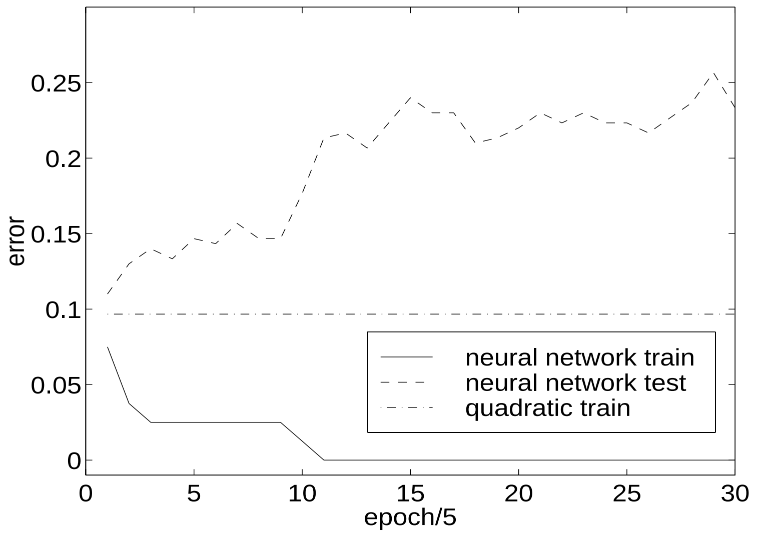


Figure 3.11: *This figure shows the error curves for the 2-20-1 neural network and the quadratic classifier. For the neural network both the generalisation error (dashed line) and the learning set error (solid line) are depicted.*

data. This is also indicated by the nonlinearity $\mathcal{N}$. A high nonlinearity is likely to indicate a poorly generalising classifier and is expected to have a large capacity. In order to be able to separate a data set with a certain error, however, a certain amount of nonlinearity is required

In this section experimental results will be presented in which we tried to relate the VC dimension and $\mathcal{N}$ with respect to a data set using the SVC and a neural network. The type of support vector classifier used, were polynomials with a varying degree. They were applied to both the 'Banana' and 'Overlap' set. The computation of a polynomial SVC of degree d is equal to calculating a linear decision function in a $d(d+3)/2$-dimensional space (see chapter 2 and [CV95, Vap95, Sch97]). From the theory [Vap82, Vap95] it is known that a linear decision function in an n-dimensional space has a VC dimension of $n+1$. Therefore, a polynomial SVC of degree d has a VC dimension of $(d^2+3d+2)/2$. For increasing d the nonlinearity should also increase. In our experiment we used the following degrees for the support vector classifier: $2, 3, 4, 6$ and 10. Their corresponding maximal VC dimensions are: $6, 10, 15, 28$ and 66. Note that the actual VC dimension may be much smaller due to the small number of support vectors used. It was therefore expected that especially for experiments with the 'Overlap' set the influence of the capacity and nonlinearity is the largest due to the large overlap and small sample size.

On both sets also a neural classifier was trained using the LM learning rule and having the same number of hidden units as there were degrees for the SVC, i.e. 2, 3, 4, 6 and 10. The calculation of the VC dimension of such a neural network classifier is more difficult. From

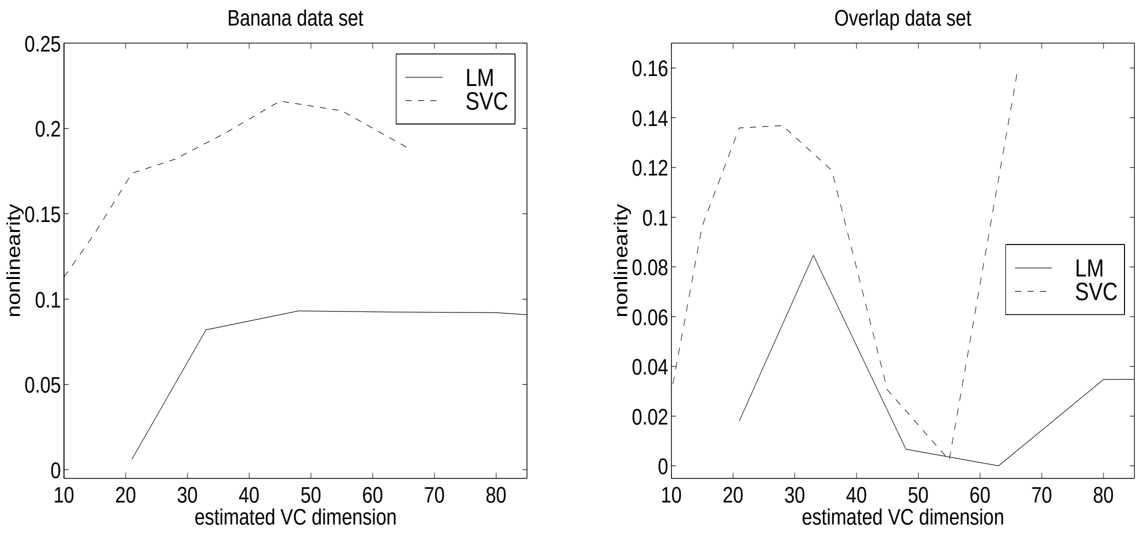


Figure 3.12: *The left graph shows the nonlinearities for the SVC and neural classifier on 'Banana', and the right graph on 'Overlap'.*

the theory [Bau88, BH89] it is known that the VC dimension of a neural network is bounded as follows:

$$2n\lfloor\frac{h}{2}\rfloor \quad \leq VC \leq \quad 2(h(n+2)+1)\log(e(h+1)),$$

where n is the dimensionality of the feature space, h is the number of hidden units and e the base of the natural logarithm. In the experiments we estimated the VC dimension of the network as the rounded average of the upper and lower bound. This resulted in the following estimations (per number of hidden units) of the VC: 21, 33, 47, 80 and 136.

Since the 'Overlap' set has a quadratic function as (best) decision function, the (best) nonlinearity value is also known and could therefore serve as a reference. In figure 3.12 the nonlinearity curves have been depicted for the two data sets. The left graph depicts two nonlinearity curves for the 'Banana' set and the right graph for the 'Overlap' set.

From these figures it is clear that an increasing VC dimension, e.g. increasing degree of the polynomial, the nonlinearity also increases. This means that the classifier is adapting too much to the data and has less generalisation capacity, which was expected. However, the $\mathcal{N}$ for the SVC increases faster and larger than the $\mathcal{N}$ of the neural classifier. The right graph (figure 3.12) shows a break down situation of the SVC algorithm. At VC$= 55$ (degree 9 of the polynomial) the nonlinearity reduces to zero. A first glance at the graph would suggest that due to the high dimensional space the SVC cannot find a suitable solution. A closer

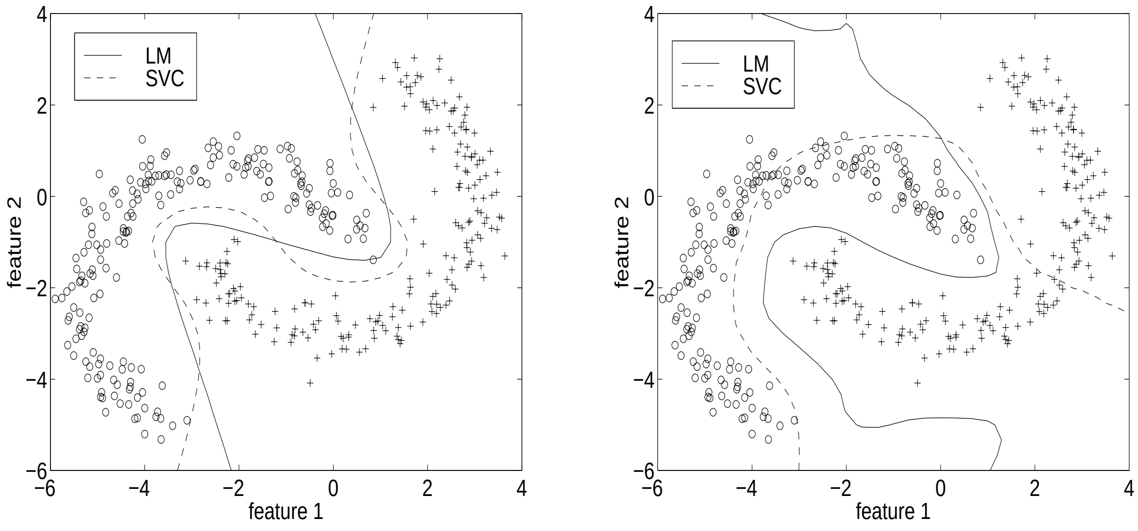
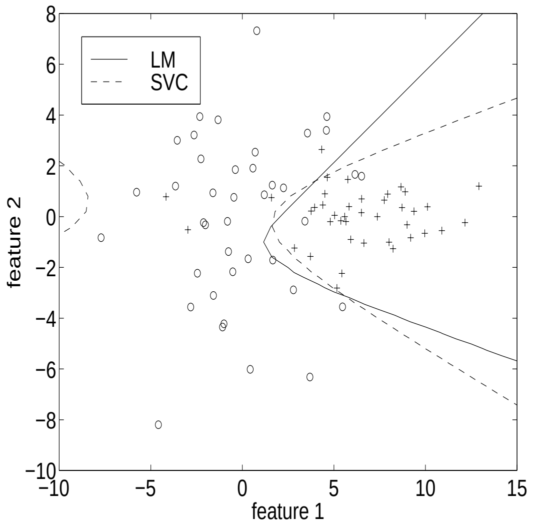


Figure 3.13: *The left figure shows the best decision functions for 'Banana'. The right figure shows an 'overtrained' situation, the degree of the polynomial is too large.*

inspection, however, shows that the matrix $\mathbf{D}$ in equation 2.12 has numbers which exceed the computational accuracy of the computer. Results beyond VC$= 55$, at least for the SVC, should therefore be qualified as unreliable.

The next two graphs in figure 3.13 illustrate that behaviour by showing the "best" and "overtrained" decision functions for the two data sets. In the left graph of 3.13 it can be observed that the SVC shows less overtraining than the neural network. The support vector classifier is more smooth and follows the boundaries of the classes better than the network with three hidden units. The second picture (on the right in figure 3.13) shows the neural network with 10 hidden units (with an estimated VC dimension of 136) and a polynomial of degree 10 (SVC with an estimated VC dimension of 66). In this case the SVC is "out of control", probably the dimensionality in which the classifier is calculated is too high. This causes an overtrained SVC which does not classify the data anymore in a correct way (almost 50% error). On the average "dips" for the neural network as shown in figure 3.12 will not occur. Figure 3.14 shows the "best" and "overtrained" decision functions for the 'Overlap' data set. Although the SVC breaks down for degree 9, it still seems to implement a reasonable decision function (figure 3.14 on the right).

Figure 3.15 shows the results, for the 'Banana' and 'Overlap' data set when taking the average over four experiments. The figure shows no local minimum for the LM trained networks, the observed "dip" in figure 3.12 (right) has been averaged out. This mainly due to the fact that the SVC algorithm always results in the same solution, whereas neural network



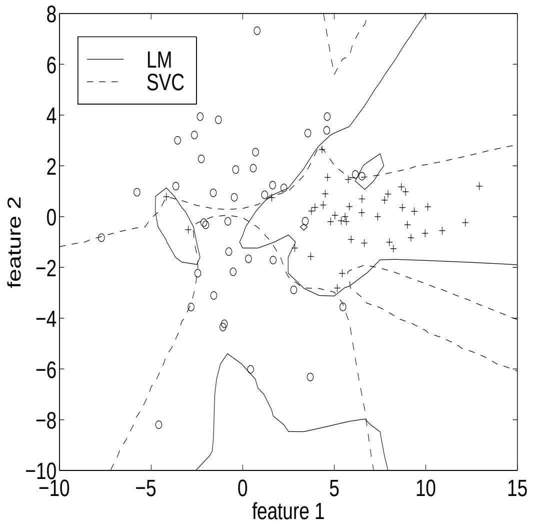


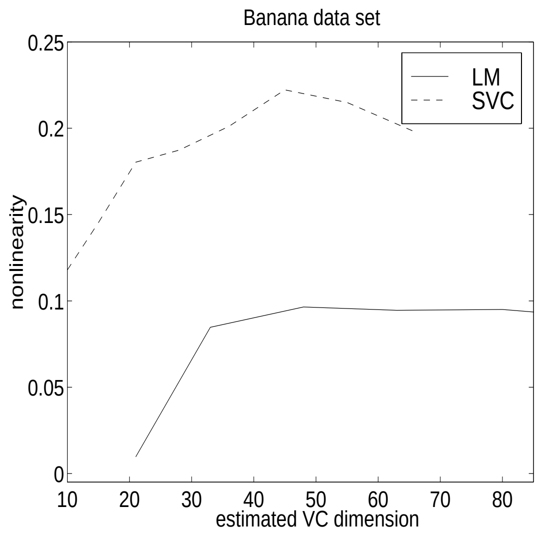
Figure 3.14: *The left figure shows the decision functions for the best classifiers on the 'Overlap' set. The right figure shows an 'overtrained' situation.*

solutions may differ due to initialisation effects. Furthermore, neural networks appear to be far more robust to high dimensional feature spaces that the quadratic optimisation algorithm used to compute the SVC.

3.5 Discussion

In this chapter the nonlinearity measure $\mathcal{N}(\mathcal{L}, \mathcal{S})$ was presented, which can be used to study the generalisation behaviour of a (neural) classifier. By comparing the nonlinearity of different classifiers, it can be detected if a network is not generalising anymore. This nonlinearity value has a close relation to the VC dimension. The VC dimension gives a bound on the capacity of a classifier but does not take the data into account as the nonlinearity does. It is related to the shape of the classifier and is affected by only those samples that influence the decision boundary, hence its relationship to the support vector classifier. It is observed that the nonlinearity of the polynomial SVC is larger than that of a neural classifier. Consequently, a network is able to implement a best decision function in terms of generalisation. Experiments in section 3.4 also showed that a larger capacity leads to a larger nonlinearity.

An algorithm was designed to avoid the computational complexity of the definition for $\mathcal{N}$. This algorithm estimates the nonlinearity using a Monte-Carlo technique. It is in its current implementation still quite slow and needs further refinement, for instance, by only taking into account the samples which lie close to the decision boundary. Only these samples influence the nonlinearity, all others add none. This can be done by creating some kind of



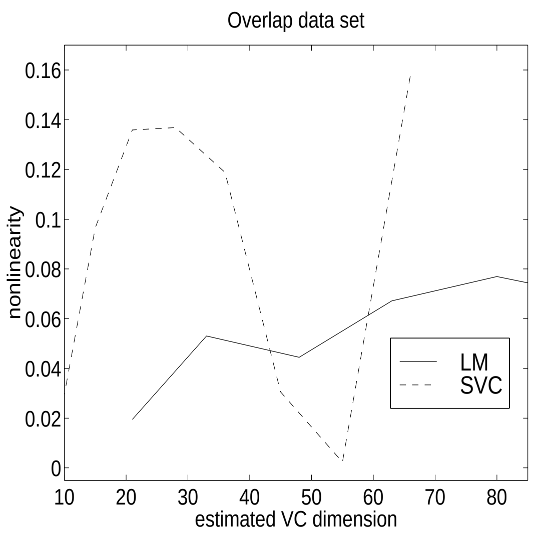


Figure 3.15: *The $\mathcal{N}$ results for the 'Banana' (left) and the 'Overlap' (right) data set after performing four experiments and averaging the results.*

neighbourhood around the current decision function (cf. the margin of the SVC). Within this neighbourhood samples are expected which affect the nonlinearity. A problem, however, may be that there are few samples within a margin and therefore little support and that non-contributing samples do influence the average. This problem has not been solved yet, but the concept and current implementation of $\mathcal{N}$ have demonstrated their use.

In general the nonlinearity measure seems to be a useful mechanism to monitor the nonlinearity behaviour of a (neural) classifier, although in practice it is not known a priori what the bound on the nonlinearity should be. This can be circumvented by taking a highly nonlinear classifier, i.e. a *1* nearest neighbour classifier and using its nonlinearity as an upper bound. This way a (neural) classifier will always be less than the "worst" classifier (e.g. a *1*-NN). However, when the dimensionality of the data is increased and its intrinsic dimensionality is fixed, the nonlinearity appears to decrease. This mean if noise is added to a feature vector by adding a noisy but useless dimension, the classifier's nonlinearity with respect to the data appears to decrease since the classifier has to be re-trained in a higher dimensional feature space. In this space the samples may be more easily separated and hence lead to a "simpler" classifier. For example, three samples in a 1-dimensional space which may require a quadratic decision function can be separated using a linear function when an extra dimension is added. According to definition 4 this will lead to a decrease of nonlinearity. Note that if the classifier remains the same, no change in nonlinearity will occur. Some preliminary experiments confirm this, although, they are not reported here.

Chapter 4

Classifier redundancy

4.1 Introduction

From the literature it is known that using a large number of hidden units is beneficial in network training [HKP91, HSW89]. Using a large number of hidden units causes the network to reduce its error much faster than if it would have the optimal number of hidden units, i.e. optimal in the sense of the smallest number that is needed to solve the problem. For example, the XOR problem can be solved using a feed forward architecture having two hidden units. This is in theory the optimal architecture. In practice, however, using five units causes on average a much faster convergence. This leads to the problem of what happens during training with the hidden unit(s) in a network. If one is able to understand their behaviour, one might be able to select the best architecture for a particular problem. That is, it should have not too many hidden units since such a network takes a long time to train and not too few otherwise it might not learn at all. The aim of this chapter is to **study** the behaviour of the hidden units during training. Parts of this chapter have been published earlier in [HDK97] and [HD97].

Figure 4.1 shows a typical training cycle of a neural network as viewed through the mean squared error (MSE) on the training set. It shows some sudden changes of the error, indicated by the arrows in figure 4.1. These changes have been studied by several people [Sch93, Ann94, Vog94] and it was discovered that at those "changes" the network *breaks its symmetry*. That is, hidden units start to behave "differently". When network training is started, the weights $\{w_{ij}^{(1)}, w_{ij}^{(2)}\}$ are initialised to some small random values around 0. For a feed forward network with sigmoidal units, the type of network studied in this thesis, this means that all the (hidden) units have approximately the same output (somewhere around 0.5) and will therefore have approximately the same direction in their decision functions. In figure 4.1 (right) this situation at the initialisation point of the network is depicted. The figure shows the decision functions of 25 hidden units from a 2-25-1 neural network. Note

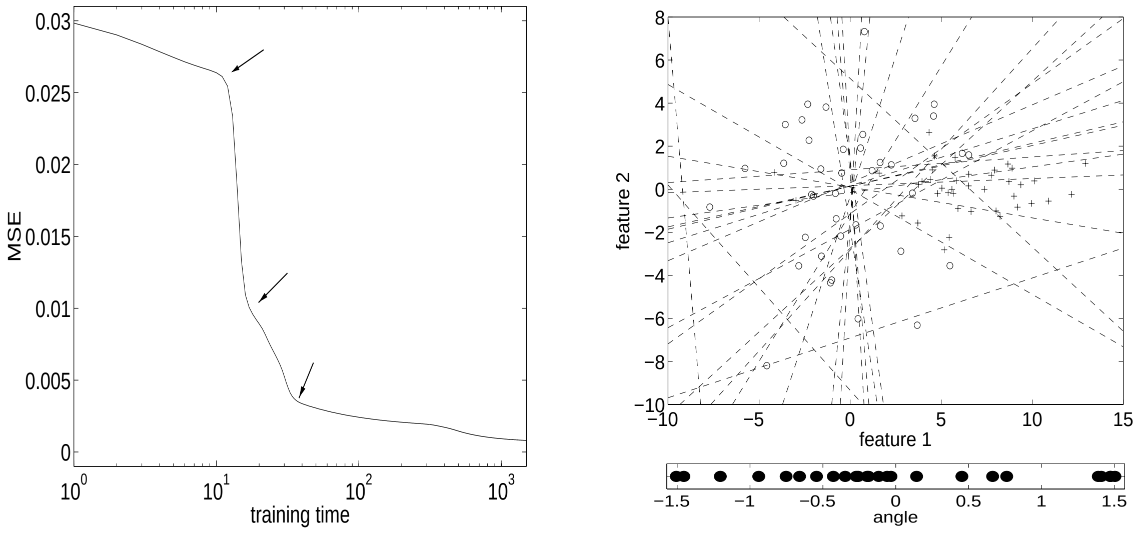


Figure 4.1: *In the left figure a typical training curve when training a feed forward neural network is shown. The right figure shows the decision functions of each of the hidden units at the initialisation phase. Their clustering behaviour is shown in the angle plot (see text).*

that these hidden units are derived from a single layer architecture and therefore implement a linear decision function. Below the picture of the decision functions their relative angles have been plotted. This angle is determined by calculating the slope of a decision function with respect to the “feature 1”-axis. Due to the fact that the weights are small, the decision functions tend to cluster in the hidden unit space, i.e. form clouds of dots in the angle figure. Shortly after the initialisation, step these clusters are broken into (sub)clusters, due to the fact that a small change in the initial weights causes a major shift in the output caused through the use of the non-linear sigmoid function.

As stated, it is expected that during training groups of hidden units are clustered, e.g. as in the example shown in figure 4.1. This can be visualised by imagining the hidden units in some high dimensional space in which they follow some trajectory. Initially all trajectories are clustered together as the units have almost the same weights, but it is expected that due to training the trajectories start to deviate. The space in which the hidden units are represented will be referred to as *hidden unit space* and is a space which is spanned by the weights of the hidden units (see section 4.3 for more details). The way they move around in the hidden unit space is determined by the learning algorithm, the data to be classified and the number of available units. Also the training time has to be taken into account, since longer training means that a larger part of the hidden unit weight space can be “explored”. The maximum number of clusters that may result in the training process is determined by the number of hidden units present, the number of clusters will not exceed this number, and the training time. This final number may, therefore be influenced by a possible overtraining of the network. Furthermore, the hidden units "construct" the decision function of the network. Therefore the number of hidden units is related to the nonlinearity as introduced in chapter 3.

This chapter will focus on how the hidden units can be followed during training. The assumption that units are clustered, suggests the use of clustering techniques. In the existing literature numerous methods are available which can be used to detect clusters in the data. However, we also want to have a visual interpretation of the space in order to detect the breaking of the clusters and to be able to follow the units in their space during learning. This requires a method that is able to do both, clustering and mapping a high dimensional space onto a (preferably) 2-dimensional space. Section 4.2 will introduce some basic clustering techniques which could be used to analyse clustering behaviour, and it will introduce the self-organising map (SOM) which performs both clustering and mapping. The next section will deal with the visualisation of the mapped space. After training a SOM, its results have to be interpreted in order to perform the visualisation. Section 4.4 will show the experiments which have been performed to investigate the behaviour of hidden units on various problems using different learning rules. The chapter will end with a discussion of the results.

4.2 Clustering techniques

This section introduces some basic clustering techniques which can be used to study the clustering behaviour of the hidden units. Most of these techniques, however, do not enable the visualisation of the the trajectories such units make. They are only useful for the detection of clusters of units. The advantage of these techniques is that they are usually quite simple to perform and implement although some are computationally expensive. The last clustering technique, the self-organising map (SOM), can be used to do both, clustering and visualisation.

4.2.1 Isodata/k-means clustering

This technique is based on randomly choosing k initial cluster centres, or means. These initial cluster centres are updated in such a way that after a number of cycles they represent the clusters in the data as much as possible.

A drawback of the k-means algorithm is that the number of clusters is fixed; once k is chosen it always returns k cluster centres. The Isodata algorithm circumvents the problem by removing "redundant" clusters. Whenever a cluster centre is not assigned enough samples,

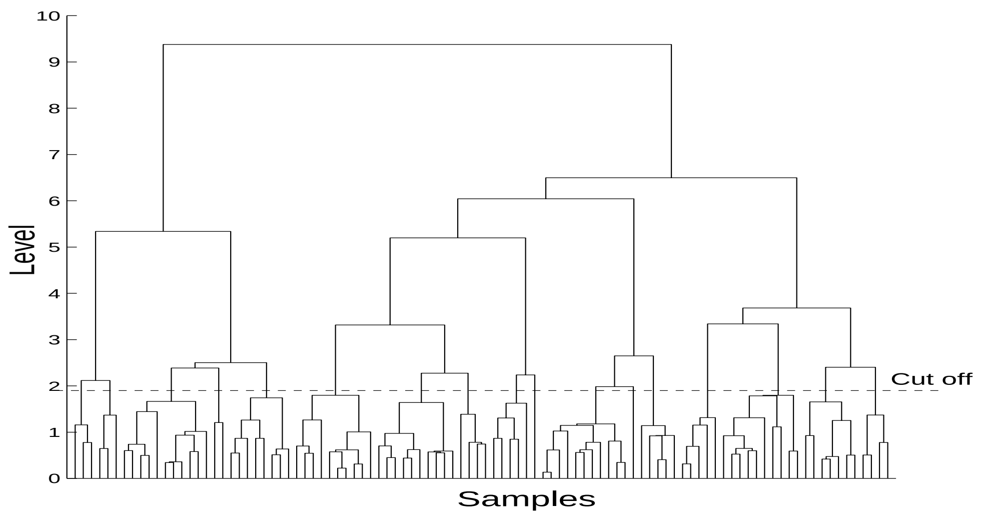


Figure 4.2: *This figure shows an hierarchical clustering performed on the 'Banana' set. The bottom row shows the samples as individual clusters, the top row the final single cluster. The ordering on the horizontal axis is a convenient one and induced by the clustering method.*

it may be removed. In this way one is left with a more or less optimal number of clusters. The problem of choosing the initial number of clusters still remains, but by taking k large enough this will usually be no problem.

Vector quantisation [Koh89, Bac95] is a method which is comparable with the k-means method. The vector quantisation method also takes a number of initial cluster centres as a starting point for finding the clusters. In contrast to k-means clustering, however, one sample at a time is used to change a cluster centre, referred to as a *prototype* vector.

4.2.2 Hierarchical clustering

Another way to perform cluster analysis is to create a tree like structure, i.e. a dendrogram, of the data under investigation. By specifying a distance measure between the samples in the data set, e.g. Euclidean distance, a tree can be made which shows in which order samples are collected. One starts in this algorithm by taking each sample as a single cluster. These single clusters can become connected in a higher level by using "linkage". This linkage determines how clusters are connected to form a new one, and what the new cluster distances will be. For instance average linking takes two cluster centres which lie closest to each other and sets the new cluster centre to the average of the previous two. Finally a single cluster representing the whole data set remains. Figure 4.2 shows a dendrogram, or tree, on the 'Banana' set (see section 3.3 and figure 3.2 for a description).

As can be observed from figure 4.2, it is unclear how samples are clustered. However, by labelling the bottom part of the dendrogram it becomes clear how samples are connected. If one performs an Isodata clustering on the pictured data set, the number of resulting clusters is 17. This means that the dendrogram should be cut off somewhere near level 2 (the dashed line in figure 4.2) to obtain the same number of clusters. In practice it is hard to determine that number. Furthermore, the dendrogram does not give any insight in how the data space looks. It is not possible to map the clusters onto some 2-dimensional space that shows the clusters. This is a major drawback of this method and the previously shown methods (e.g. *k*-means). They do, however, give insight into the number of clusters present in the data set and their distances.

4.3 Feature space visualisation

The self organising map (SOM) is able to do both: extract cluster centres and perform a high dimensional to 2-dimensional mapping ([Koh89, Koh90, RSM91, BLV90]). However, the trained map is not directly available for displaying the nature of the high dimensional space. Kraaijveld et al. [KMJ95] have introduced a method for displaying the final mapping made by the SOM. Not the actual prototype vectors (the weights of the units in the grid) in the map but their distances to each other will be displayed. This method results in a gray value image of the inter-unit distances. The value of each pixel, i.e. a SOM unit, is determined by taking the largest distance it has with respect to its direct neighbours. This means that such a gray value image is a worst case approach in displaying the feature space since the distances of a unit to its direct neighbours differ. It is thereby a safe approach since it guarantees that the distance between two neighbouring cluster centres will never be larger than the maximum distance. Figure 4.3 illustrates, through an artificially constructed example, how the method works.

The leftmost picture shows a 2-dimensional data set on which a SOM, consisting of 4×4 units, is trained. In the middle figure the trained SOM is overlayed onto the data set to show how the mapping has been performed. A SOM unit is denoted by the fat circles and their connections by the dashed lines. By using the inter-unit distances (the length of the dashed lines in figure 4.3), the rightmost figure has been created. In this picture, each pixel represents a unit of the SOM grid. The gray level of such a pixel is determined by the largest distance. Consider units 1 and 2 in the middle picture (figure 4.3). Unit 1 has small distances to its neighbours, therefore the gray value representing the distances of unit 1 will be dark (almost black). The other unit (unit 2) has both large and small distances to its neighbours. Since we take the largest distance as gray value, the pixel representing the distances of unit 2 will be high (almost white). In general, the smaller the (maximum) distance of a unit to

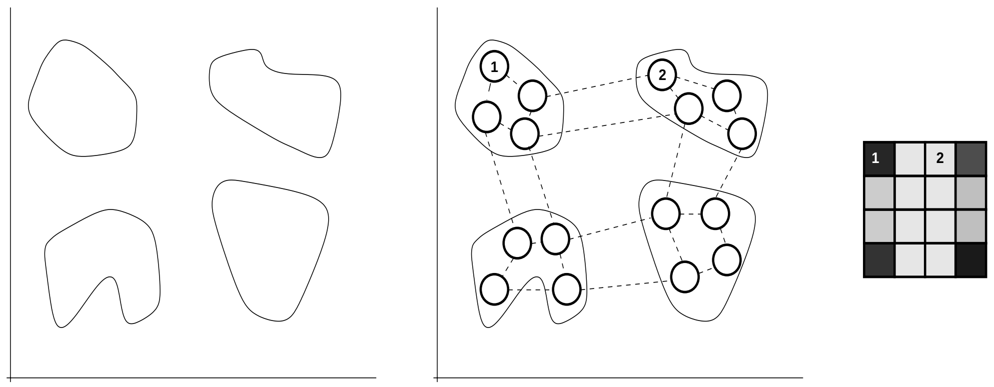


Figure 4.3: *This picture illustrates how the visualisation method works for an artificial 2-dimensional example. From left to right: the data set, the data set on which a SOM was trained and the visualisation of the trained SOM.*

one of its neighbours the darker its corresponding pixel will be. This mapping method will be used to study the hidden unit weight space.

Figure 4.4 shows an example of a 3-dimensional data set and the corresponding gray value image of the SOM after training. The 3-dimensional data set (formula for generation derived from [KMJ95] and trained using the SOMPAK package [KHKL95]), clearly shows a separation between the two clusters. Therefore, after training a SOM and corresponding post processing the gray value image should display that property of the data set. The gray value image indeed shows a bright area in the map which corresponds to the void in the original data space. Since there is a significant distance between the two clusters, the resulting image shows a bright white, i.e. high, line. The method of displaying makes use of the fact that the SOM is topology preserving.

How can the hidden units be followed in their space during training? A first step is to collect the weights of the hidden units during various training sessions. It is assumed that the feed forward network consists of a single hidden layer. Furthermore, it may have an arbitrary number of input units and output units. A hidden unit can therefore be represented in a $(N + O + 1)$-dimensional space, assuming N inputs and O outputs. Figure 4.5 shows how the weights are derived from a 2-2-1 neural network and encoded into a 4-dimensional vector. For each hidden unit such a vector is derived in order to follow the hidden units. In the experiments only networks consisting of 2 input units and 1 output unit were studied.

This space in which the hidden units are represented, which is sampled by training different networks and storing all the hidden unit weights during training, is then mapped onto the 2-dimensional grid of the SOM. After training, a gray value image can be created using the technique described in this section. If the different training sessions of the networks are

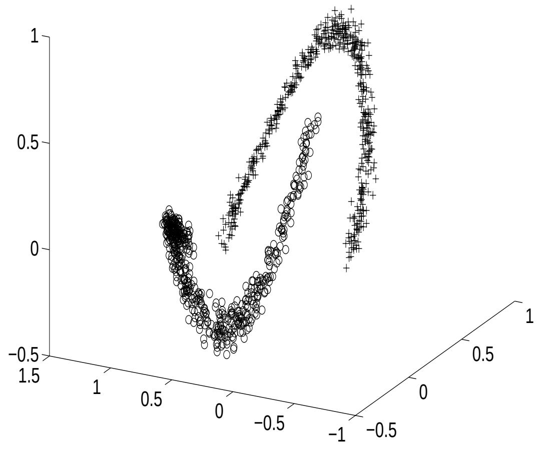
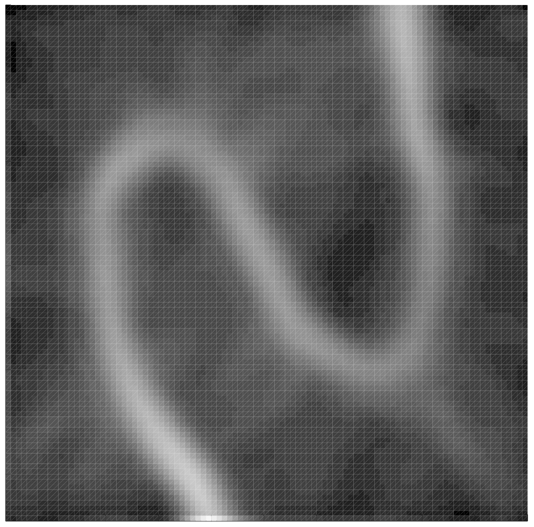



Figure 4.4: *The left picture shows the data set in the 3-dimensional space which is mapped onto the 2-dimensional SOM grid. The picture on the right shows the gray value image which is obtained by taking the inter-unit distances after training.*

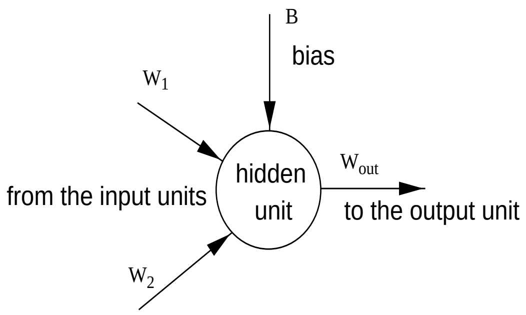


Figure 4.5: *The vector obtained from the network for training the SOM is given by (W_1, W_2, B, W_{out}). For each hidden unit such a vector is obtained.*

plotted on the map, the "route" of the hidden units can be followed. The next section will go into the details of the results obtained from these experiments.

4.4 Experiments

In order to investigate whether the hidden units in the hidden unit space are really clustered and these clusters break apart at specified points in the training process, indicated by the MSE changes, several experiments were performed. Furthermore, it was also investigated whether it is possible to detect cases of overtraining. Two different data sets were used. The first one was introduced by Annema [Ann94] (also used by Vogtländer [Vog94]) and is perfectly separable. The second data set was the 'Overlap' set (figure 3.2 in section 3.3). It was used to force a network into an overtrained situation due to its overlapping nature. Figure 4.6 (left) shows the 'Annema' data set. The influence of the learning algorithm on the symmetry breaking nature was also studied. For instance, does the speed with which the network learns, due to a faster learning rule, imply faster symmetry breaking? We restricted ourselves to the following three algorithms:

- Standard backpropagation (SB) with momentum implemented in ANN / SPRLIB [HKdRS96].
- Levenberg-Marquardt (LM) learning rule as implemented in Matlab's neural toolbox [DB93] and PRTools [Dui95a].
- Fast backpropagation (FB) as implemented in Matlab's neural network toolbox and PRTools.

The last learning rule is of special interest since it quickly brings the network into an overtrained situation (see chapter 3). In combination with a large number of hidden units it may lead to complete symmetry break or no symmetry break. For both problems the optimal number of units is two. The 'Annema' data set can be perfectly separated using two linear decisions, a horizontal and a vertical one. In a neural network they can be implemented using two units. For the 'Overlap' data set it holds that it can be best separated using a quadratic decision function (see chapter 3) which can also be implemented by a network consisting of two hidden units [HKP91, Bis95].

The network under investigation is a standard feed forward neural network consisting of a single hidden layer with 6 hidden units, in order to ensure enough redundancy. Furthermore, both problems are 2-dimensional and 2-class. Thus the network has two input units and a single output unit. During training, at each time stamp the weights of the network were saved. For the network trained in ANN / SPRLIB the time stamp was set to every 10 epochs and in Matlab after each update step. The weights of the hidden units are encoded in a 4-dimensional vector, 2 weights from the input units, a bias weight and a weight to the output unit (see section 4.3 and figure 4.5). Consequently, the hidden units are expected to 'move' around in a 4-dimensional space.

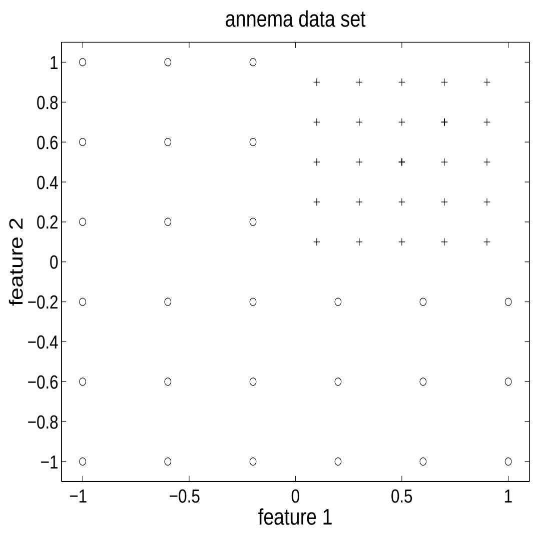
In order to diminish preferences due to initialisation, for each of the three training methods ten networks were trained. The weights of the networks stored during training were used to train a SOM. The SOMs, one for each training method and classification problem, consisted of a 2-dimensional grid of 150×150 units. This number was chosen to be large enough to ensure enough resolution in the SOM. If the number of units in the map is too small, details with respect to the symmetry breaking effect may be hidden.

4.4.1 'Annema' data set

This data set is perfectly separable using a quadratic decision function and can hardly cause overtraining of the network. The SB networks take about 15,000 epochs to train, which was the maximum number of epochs. This is due to the fact that the MSE threshold was too low. It was set to 0.001 which is never met. The LM method, however, results in a well trained network after two epochs. To some extent this is due to the nature of the software package (Matlab). The fast backpropagation method (trained in Matlab too), however, takes more steps to come to a good solution (on the average 30). The LM rule is a second order training rule which results in a much faster convergence of the network (see also section 1.4.1). It uses a modified Gauss-Newton approach [HM94, HKP91] to find the direction of the update. Since the classification problem under investigation is relatively simple a solution is obtained in only a few steps. Figures 4.6 (right) and 4.7 show the resulting SOMs for the three different learning rules. The bright areas in the figure depict regions where only a few samples have been observed whereas dark areas are regions where clusters of samples are located.

Consider the SOM image for the SB rule in the right picture of figure 4.6. It can clearly be observed that there are clusters of samples indicated by the dark areas surrounded by bright areas. This indicates that the hidden unit weights tend to end up in the same region in the hidden unit space. This can be checked visually by displaying the trajectories of the hidden units for each network in the map as displayed in figure 4.8. If one performs a k-means clustering on the final weights of the hidden units it is expected that some of the hidden units belong to the same cluster. The final weights are those weights which were obtained from the last epoch in a training cycle. An Isodata clustering, starting with 18 clusters initially, was performed on those weights of the hidden units of the ten networks. The value 18 was chosen in order to allow enough freedom for clustering.

Table 4.1 shows the results of the Isodata procedure.



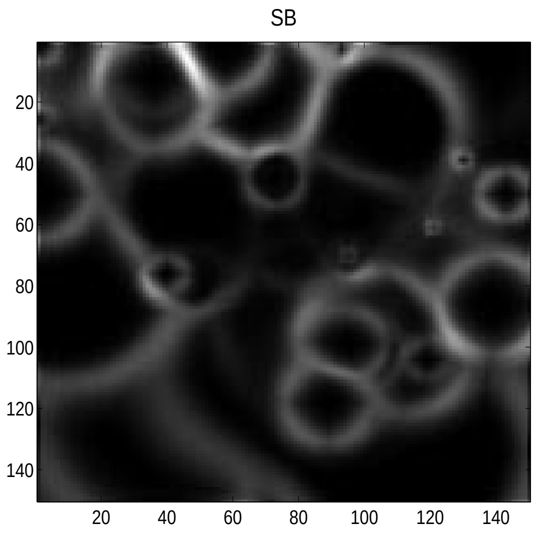
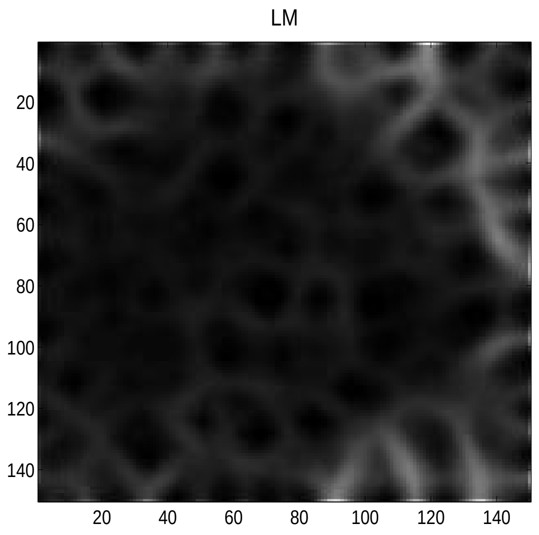


Figure 4.6: *The left figure shows the 'Annema' training set for the 2-6-1 neural network. The figure on the right shows the SOM trained on the hidden unit weights resulting from the SB procedure.*



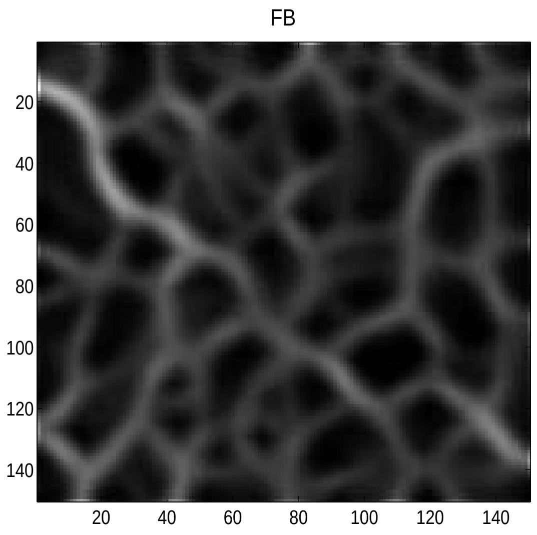


Figure 4.7: *The figure on the left shows the SOM trained on the hidden unit weights resulting from the LM procedure. The figure on the right is the result for the FB method.*

Table 4.1: *Isodata clustering, using 18 initial clusters, on the final weights of the networks trained using SB method on the 'Annema' set.*

	Cluster number														
net	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	X			X				X		X		X		X	
2				X					X	X		X		X	
3		X					X						X		X
4	X				X				X				X	X	
5		X	X										X		X
6				X		X				X	X	X			
7		X	X	X		X							X		X
8		X	X						X				X		X
9		X	X						X				X		X
10		X	X						X				X		X

In the clustering results -the following of tracks and the Isodata clustering- it can be observed that the hidden units of different networks follow roughly the same trajectory through the weight space and end up in the same cluster. For instance networks 9 and 10 clearly show this type of behaviour. A visual inspection (figure 4.8) of their trajectories indeed show that their hidden units share paths. Note that not all of the hidden units in a network end up in a single cluster. For instance network 3 has only 4 assigned cluster numbers, whereas network 1 has 6 different cluster numbers. Figure 4.8 shows the trajectories of network 9 and network 10. A visual inspection shows that they indeed have endpoints in common. Moreover, they share 4 trajectories, therefore the final weights of the hidden units end in the same cluster. This is confirmed when performing an Isodata clustering on the final weights.

Let us now consider the pictures for the LM rule in figure 4.7 (left) and the FB rule (right). Although in both figures clusters are present, the clustering in the right picture (LM method) is far more sharper. This sharpness is indicated by the bright lines in the figure. In the LM figure (figure 4.7 left) shows to be more flat. This LM figure shows less "clustering". This indicates that the LM rule quickly finds a suitable solution whereas the SB slowly walks towards a solution. When now the trajectories of the hidden units in the maps are drawn, it shows that the SB rule finds a weight solution that does not change very much and thus results in an area surrounded by "mountains" (the bright areas). In the left part of figure 4.9 this behaviour is shown for a particular realisation of a network training cycle using the

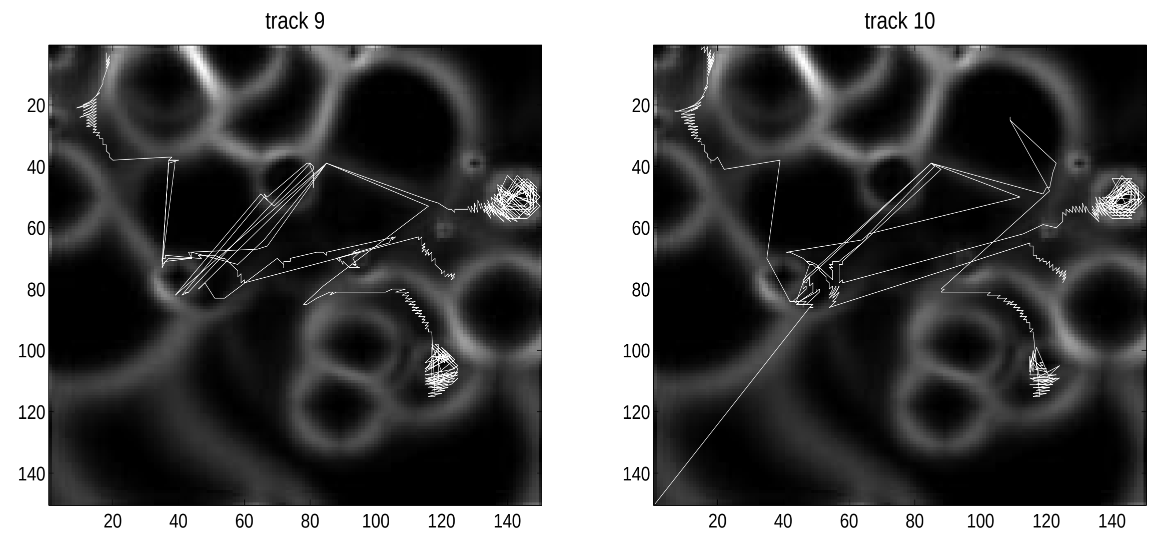


Figure 4.8: *The visual interpretation of the Isodata clustering results. On the left the trajectories of network 9 are drawn, on the right the results for network 10.*

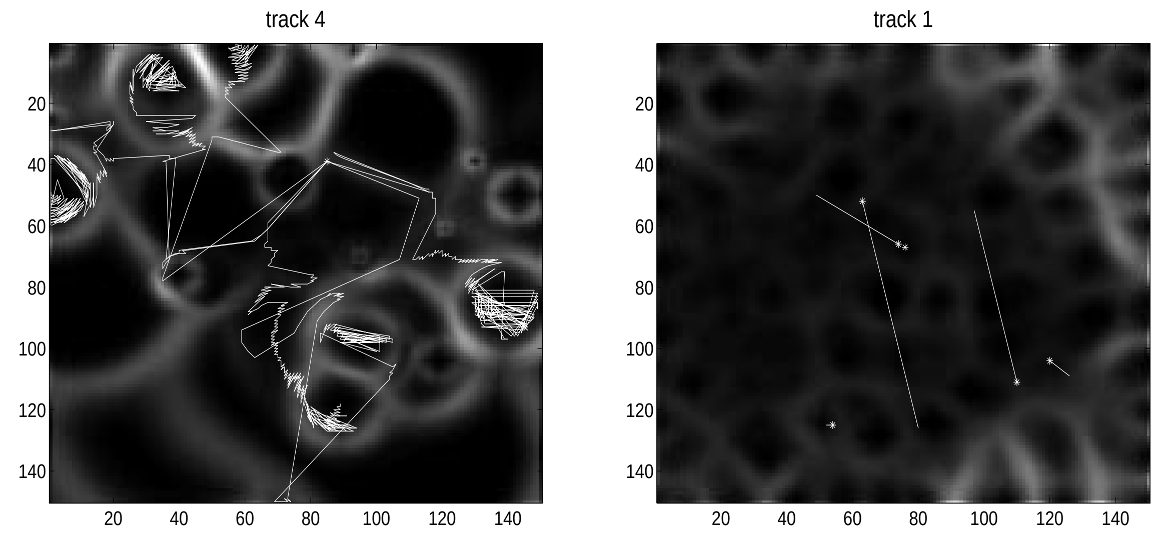


Figure 4.9: *The left figure shows the behaviour of the hidden unit weights of a particular network realisation during SB training. The figure on the right shows the behaviour of the LM rule. The symbol '*' denotes the starting points of the weights.*

SB method. The right part of figure 4.9 shows the behaviour of the LM rule. The behaviour of the FB method (see figure 4.7 on the right) is somewhat in between the LM and back-propagation behaviour. There are more clusters visible but still the weights seem to behave rather randomly as in the LM case. However, the danger exists that we are over-interpreting the figures, but when inspecting the trajectories of the FB method they are in between the LM and SB method. In the LM trained network it appears there is no clustering at all. Due to the fact that the LM rule finds a solution that fast, the hidden units have no clustering property. That is, a solution is already found in 1 or 2 steps and therefore only a few steps in the hidden unit space are performed. It does not enable a proper training of the SOM, it is too large with respect to the number of training samples (hidden unit weight vectors) and therefore largely determined by its initialisation. For overlapping data sets, however, it is expected that there is a clustering property since in these cases the network will not find a zero training set error that fast. It is expected that this will take a larger number of steps to converge due to the overlap.

There is a clear difference in the way the hidden unit space is explored when using the different learning algorithms, but what about the symmetry breaking effects in the network? Consider a specific training cycle of the standard backpropagation. Figure 4.10 shows the mean squared error (MSE) curve during the training cycle of network number 4. During this training cycle the MSE changes rather drastically at epoch 2, 3, 4, 9 and 15. Note that at epoch 9 there is a small rise in error, whereas all the other interesting points cause a decrease in error.

Figure 4.11 shows the behaviour of the hidden units in the mapped weight space. The symbols in the figure denote the points where changes in the MSE curve occur. As can be seen the units start as a single cluster (denoted by the '*') and then gradually break down in smaller clusters. The second change results in three separate clusters. As the training continues the clusters break further apart until there are as many clusters as hidden units. In general this is an undesired situation since this may indicate overtraining, assuming enough hidden units to solve the problem. Considering the classification problem, two clusters of hidden units which perform opposite functions would be desirable, since the 'Annema' data set can be separated this was. In fact at around epoch 6 the network has already found a satisfactory solution. This epoch has about 3 clusters of hidden units.

As stated in section 4.2, more classical methods can be used to investigate the clustering behaviour of the weights. The weights under investigation were obtained from the SB networks. We have applied hierarchical and Isodata clustering with k initial clusters (as showed earlier in this section) on the weights of the hidden units. The hierarchical clustering can be used to show that for a particular network realisation the weights indeed move along a path in the feature space. Figure 4.12 shows the hierarchical cluster tree for a particular network

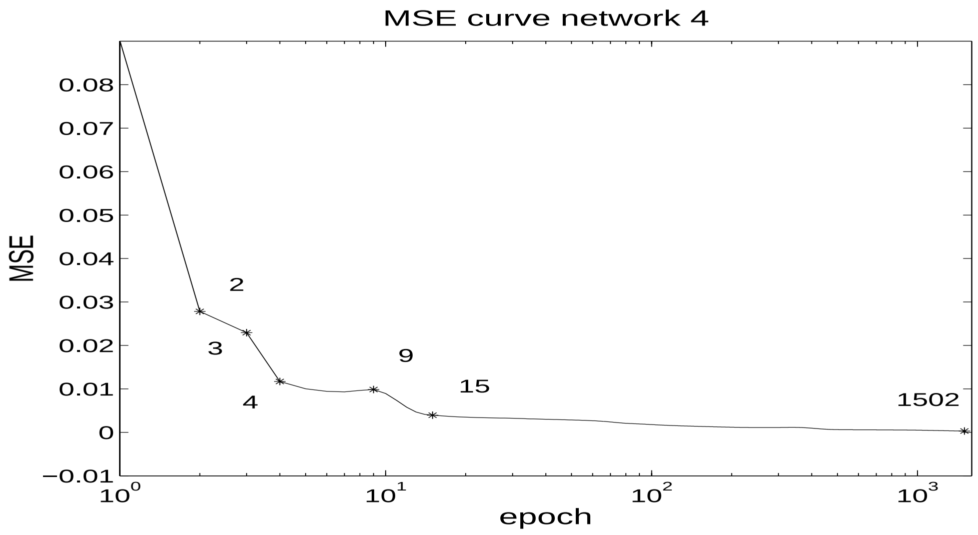


Figure 4.10: *The MSE (see equation1.8) curve for a particular training cycle of a feed forward network. The arrows indicate the symmetry breaking that occurs during training. They are also shown in figure 4.11 where the trajectories of the hidden units have been drawn. The arrows with numbers refer to the epoch numbers mentioned in the text.*

realisation trained with the SB method. The tree was obtained using single linkage, i.e. two clusters are merged which are closest and the new cluster value is the minimum distance.

The hidden units clearly follow some path through the weight space due to the fact that their weights are clustered from right to left. If the weights would not follow a path the clustering would be rather random, there would be no clear structure in the tree. In the leftmost part the clustering is not that clear; weights do follow a path but also move from cluster to cluster. This is enlarged in figure 4.13. This figure clearly shows that the weights follow a path up to a certain cluster and then move from cluster to cluster. The level at which the 6 clusters (one for each hidden unit) are merged is rather high. This indicates that there is a clear distinction between the various hidden unit trajectories.

4.4.2 'Overlap' data set

This experiment was conducted to investigate the symmetry breaking behaviour in the case of almost ensured overtraining. Here the same data set was used as in chapter 3 (see figure 3.2). This data set causes a network to be overtrained quite easily using for instance the LM training rule. The experiment was conducted under the same conditions as in the previous subsection, meaning a 2-6-1 feed forward neural network was trained with standard backpropagation with momentum (SB), Levenberg-Marquardt (LM) and fast backpropagation (FB). Figures 4.14, 4.15 and 4.16 show the mappings of the weights after training and

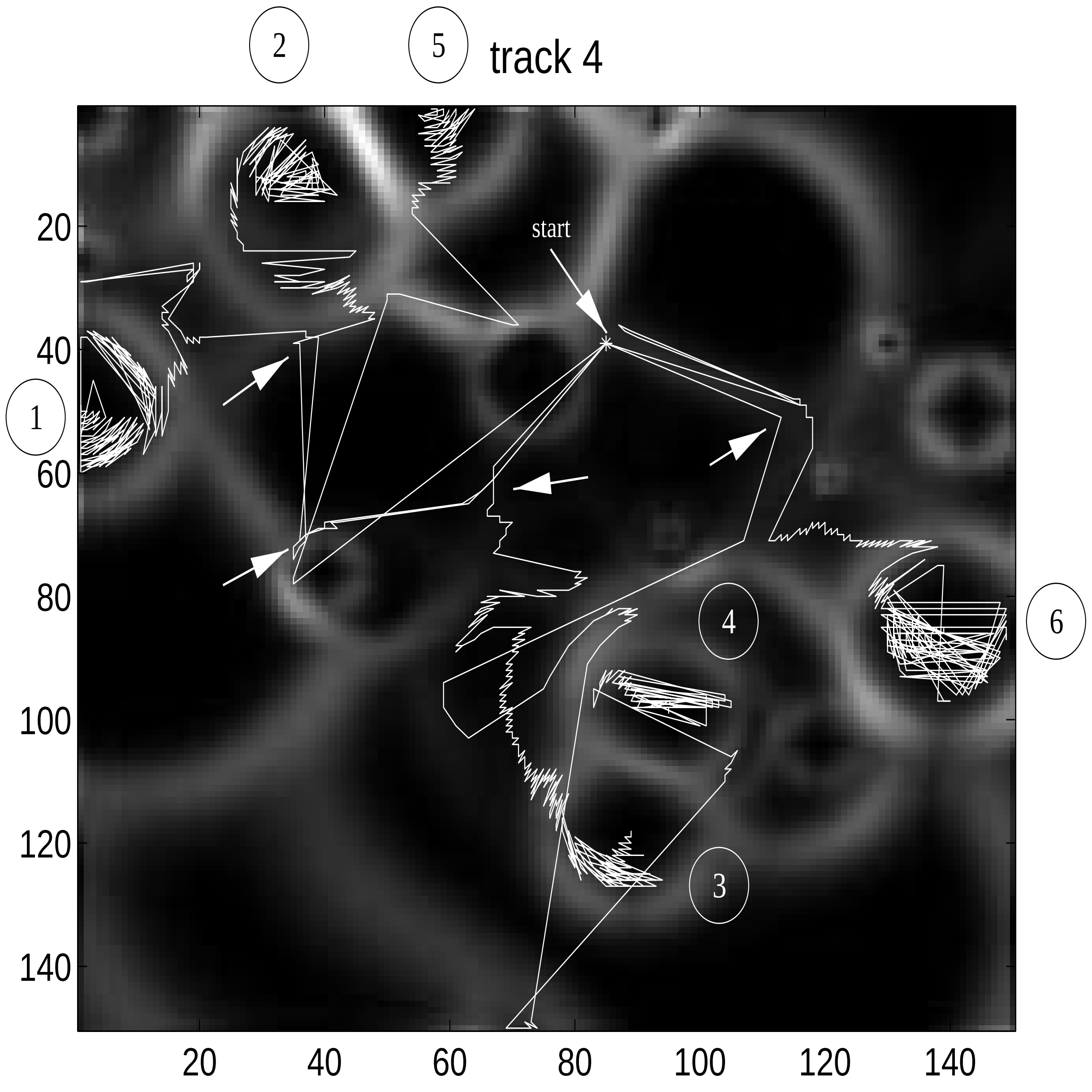


Figure 4.11: *The trajectories of the six hidden units during training of the 2-6-1 feed forward network using SB. The arrows point to symmetry breaking events which occur during training and correspond which the changes shown in figure 4.10. The numbers in the circle indicates where each unit number ends.*

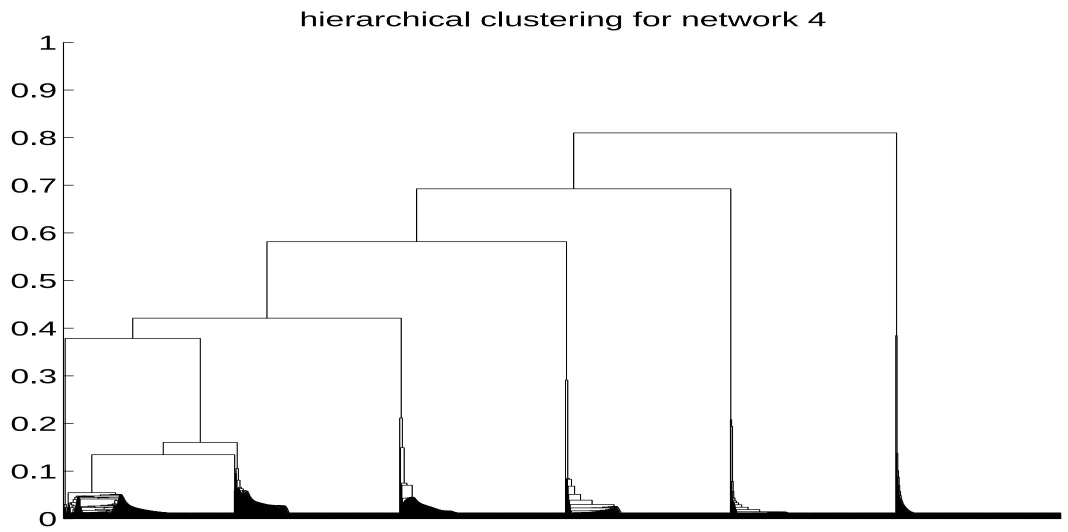


Figure 4.12: *The dendrogram of the weights of a particular network training cycle (SB trained) using single linkage. The path behaviour is clearly shown due to the right to left clustering of weights (see also figure 4.13).*

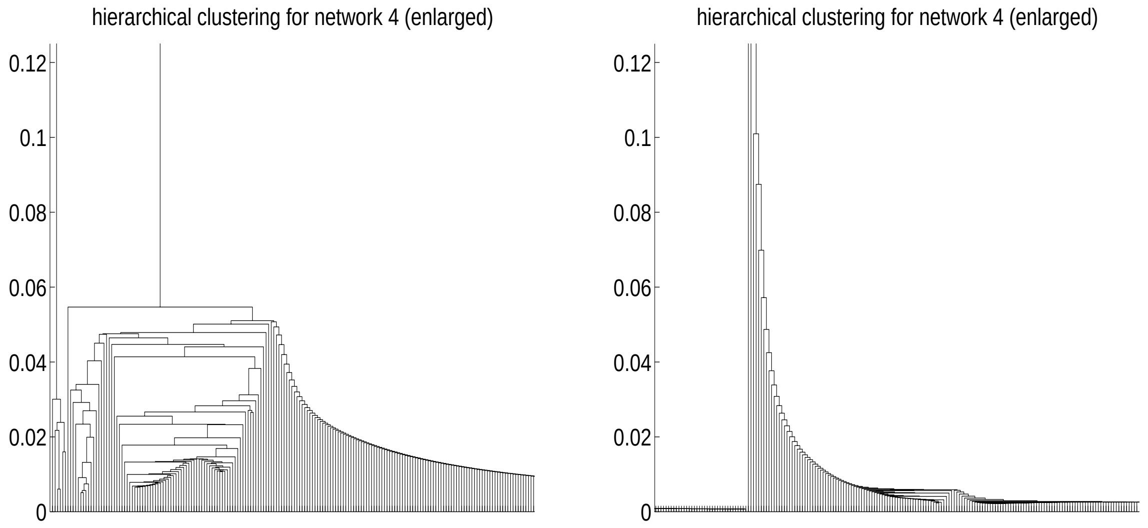


Figure 4.13: *The picture on the left shows an enlargement when zooming in on the leftmost part of figure 4.12, the right picture shows an enlargement of the rightmost part of figure 4.12.*

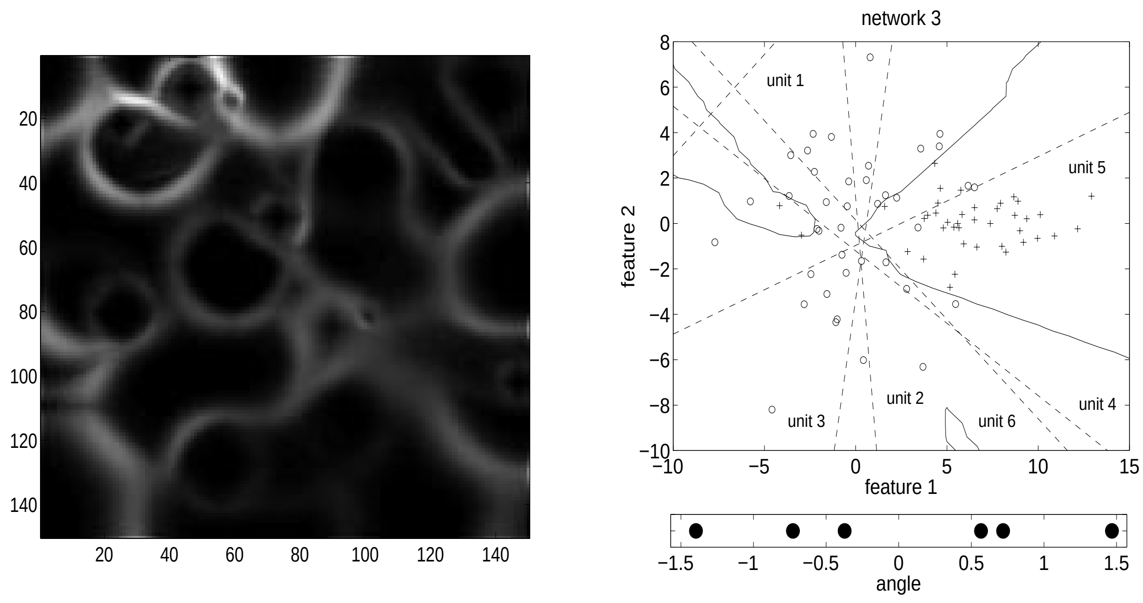


Figure 4.14: *The SOM image of the weights resulting from the SB method on the 'Overlap' set are displayed on the left. On the right the final decision function for a network is displayed. The solid line depicts the output function and the dashed lines are the decision functions implemented by the hidden units.*

the resulting decision functions from the last epoch.

The results are comparable to the previous experiment in which the 'Annema' set was used. However, due to the overlap all networks are in a severely overtrained situation after training which is clearly shown in the figures 4.14, 4.15 and 4.16 (right). The question that arises is how this overtraining behaviour affects the way the weight space is searched. If one compares figures 4.14, 4.15 and 4.16 to the ones in the previous section (figures 4.6 and 4.7) the differences can be seen. When using the FB rule the weight space has more large flat regions in the overlapping case than in the non-overlapping case ('Annema' data set). This is probably due to the Matlab implementation of the FB rule. It is implemented using an adaptive learning rate during training. A ratio term is used to limit the increase in error. If the increase in error is larger than the ratio, the weight update is rejected and the learning rate is adapted. It can be expected that in the overlapping case the increase in error behaves differently due to the overtraining behaviour. This causes large flat areas, since weight update may easily be rejected due to overlap.

Another interesting behaviour during training of a network occurs when using standard backpropagation. Due to overtraining, the MSE starts to oscillate since it cannot adapt accurately enough to the data anymore. This also causes oscillations in the weights of the

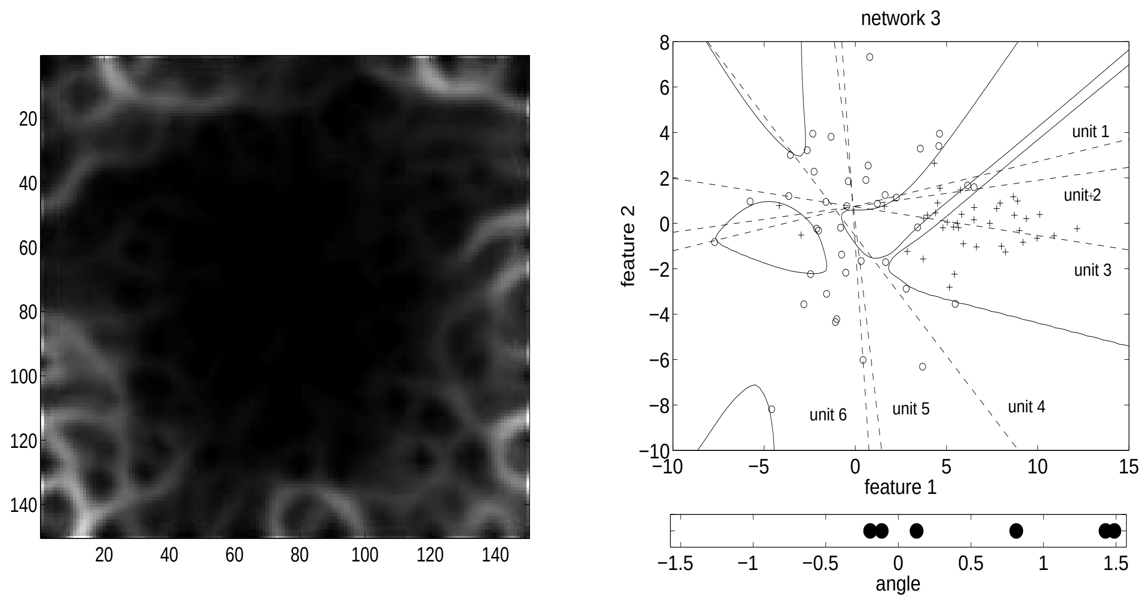


Figure 4.15: *The SOM image of the weights resulting from the LM rule on the 'Overlap' set are displayed on the left. On the right the final decision function for a network is displayed. The solid line depicts the output function and the dashed lines are the decision functions implemented by the hidden units.*

network and is clearly shown in the trajectory figure. In figure 4.17 this behaviour is shown for a particular network. After the last symmetry break has occurred the network weights start to oscillate and cause regular "jumps" in the weight space between units.

For this 'Overlap' set a hierarchical tree was built to show the clustering and path behaviour. The tree was built using the weights of the network shown in figure 4.17. In figure 4.18 the final tree is depicted. As can be observed, the paths are less "smooth" than for the Annema data set (see figure 4.12). This is due to the fact that the network becomes overtrained and shows oscillations (see figure 4.17). Therefore the clusters tend to be spread. This is in contrast to the other experiment with the 'Annema' set where the single linkage clearly shows a smooth behaviour in clustering from right to left. Figure 4.19 shows this behaviour in more detail for the leftmost and rightmost part of the tree.

In this problem Isodata clustering was also used on the final weights of the hidden units of the SB trained networks. Table 4.2 summarises the results. In contrast to the clustering in table 4.1, there are no networks that share all the clusters. Clusters, however, are shared by the different hidden units, but not as strong as in the 'Annema' case.

This may be due to the fact that this data set has an overlap and therefore allows a much

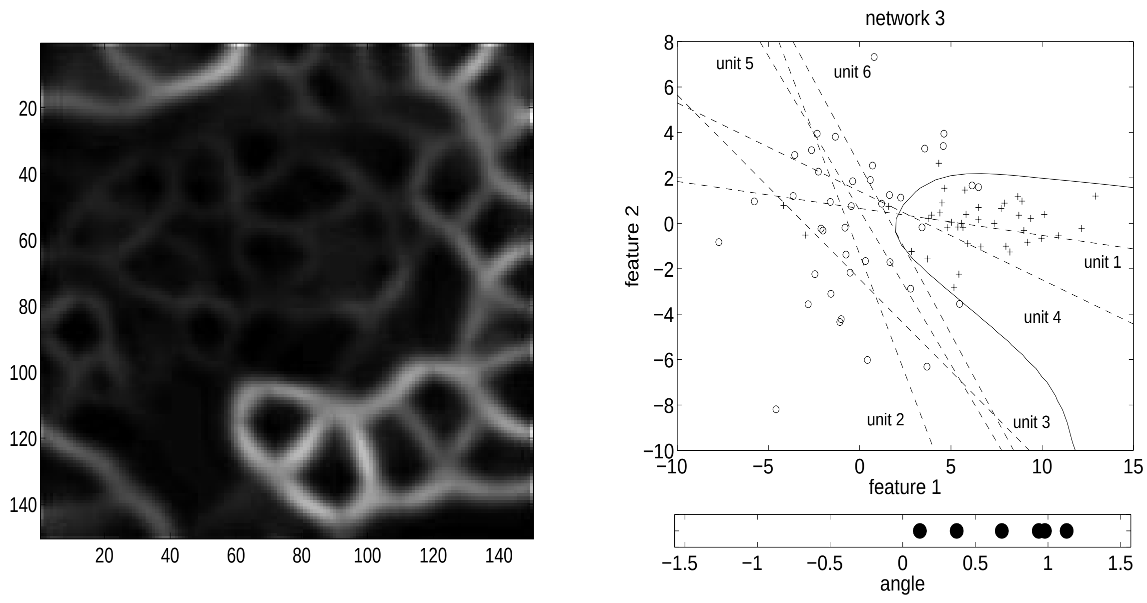


Figure 4.16: *The SOM image of the weights resulting from the FB method on the 'Overlap' set are displayed on the left. On the right the final decision function for a network is displayed. The solid line depicts the output function and the dashed lines are the decision functions implemented by the hidden units.*

wider variety of network solutions.

4.5 Discussion

We have shown how the learning behaviour of a feed forward classifier can be followed by probing the weights of the hidden units. These units follow some path through the unit weight space according to the data and training set. This behaviour can be visualised by projecting the weight space onto a 2-dimensional plane using a SOM and interpreting the resulting projection. It turns out that the weights share trajectories in the hidden unit space during training. Right after the first update step is performed, the weights break apart into several smaller clusters. Depending on the training time these (sub)clusters break into even smaller clusters until no further cluster breaking can be done. If no cluster breaking occurs anymore this may indicate a serious problem: the network might be overtrained. This overtrained situation, however, can only be detected if one knows how the different classes are overlapping, which in general is unknown. These cluster breaks are related to the changes one can observe in the MSE curve during training. This relation between the symmetry breaking, i.e. cluster breaking, and changes in the MSE curve has been already observed by others [Ann94], although not clearly shown. Another interesting phenomenon that occurred

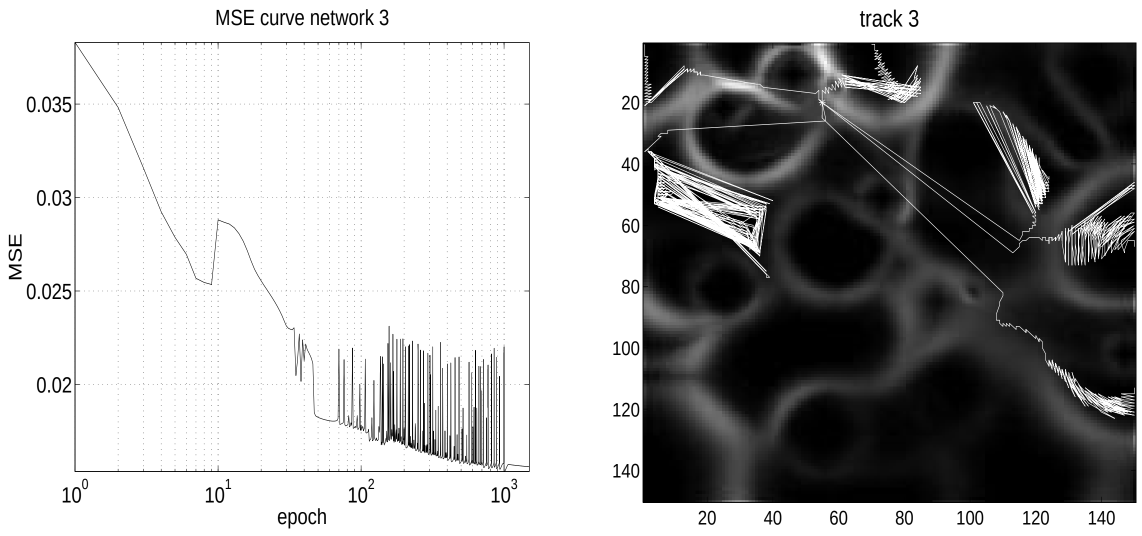


Figure 4.17: *The figure on the left shows a part of the MSE curve for a particular training session of a network using the SB rule. The oscillation behaviour of the MSE is reflected in the trajectory figure of the hidden units (on the right).*

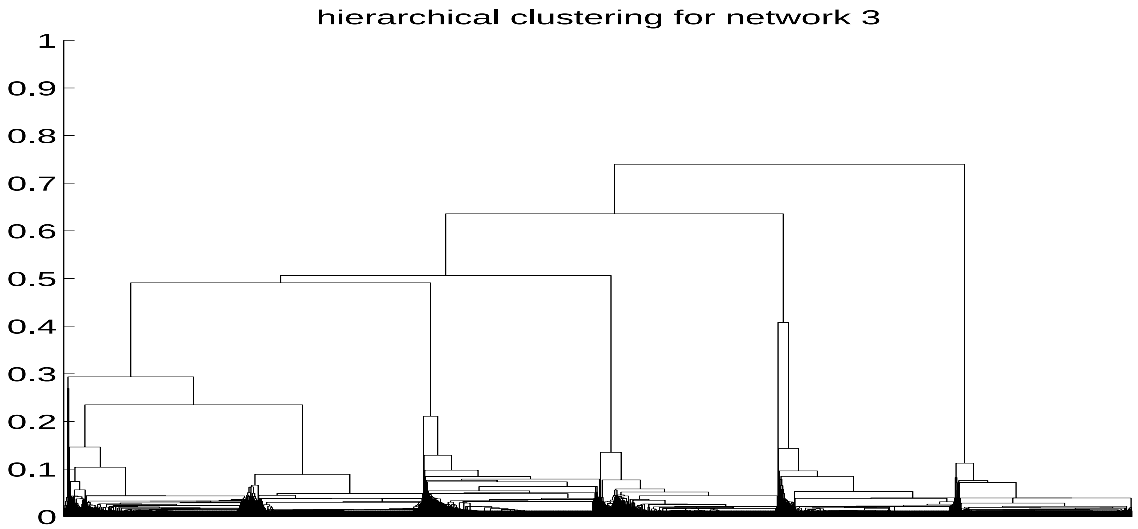


Figure 4.18: *The dendrogram for the weights of a particular network realisation using single linkage. The path behaviour is shown due to the right to left clustering of weights.*

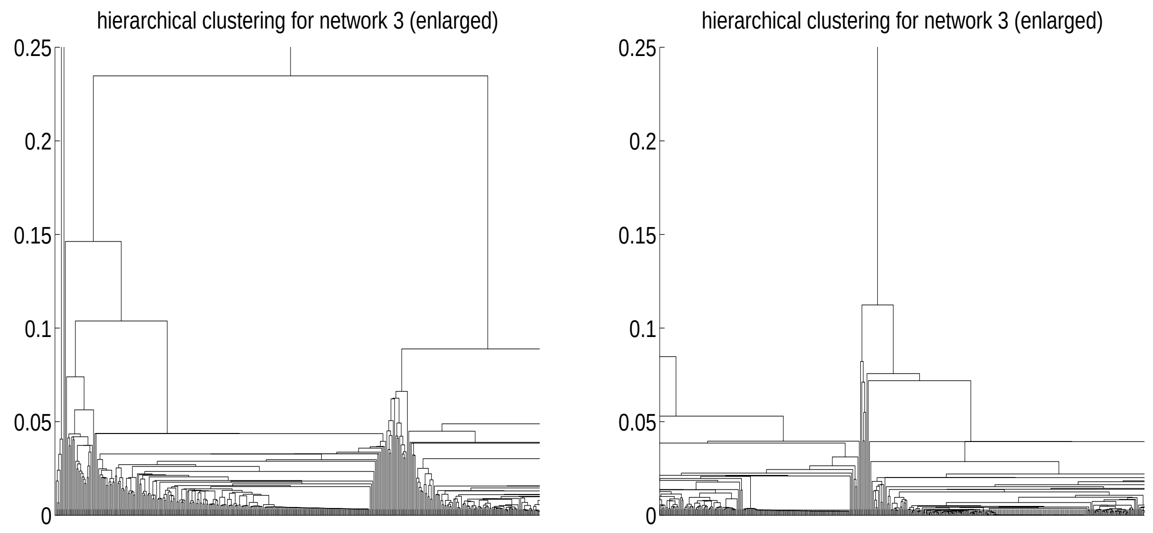


Figure 4.19: *The picture on the left shows an enlargement when zooming in on the leftmost part of figure 4.18, the right picture shows an enlargement of the rightmost part of figure 4.18.*

Table 4.2: *Isodata clustering, starting with 18 clusters, on the final weights of the networks trained using the SB method on the 'Overlap' set.*

	Cluster number														
net	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1			X				X	X						X	X
2	X						X			X					X
3					X	X		X						X	X
4		X					X				X				X
5		X					X	X			X				X
6				X			X	X						X	X
7		X			X			X				X			X
8				X	X			X						X	X
9		X		X			X	X				X	X		
10	X								X	X	X				X

is that the different network trajectories were not unique (shown by the *k*-means clustering and figure 4.8). This means that trajectories of hidden units of different networks are shared. This in turn indicates that different networks explore roughly the same parts of the weight space, although they have been initialised randomly. This is important since it indicates that the learning rule directs the network weights towards the same solution for a given set of data.

Our approach of mapping the weights of the hidden units does have a serious drawback. As we have assumed that a feed forward network was used consisting of only a single layer, our approach will not be valid for more layers. This problem may be overcome by modifying the network in such a way that only a single hidden layer remains. Another issue which has not been dealt with is the way the weights have been encoded. The reader may have been wondering what happened with the bias weight of the output unit. This bias has not been taken into account, since it only will result in an offset of the final decision function, and is of no influence in the direction of the decision function of the hidden units. A logical second experiment, however, would be to incorporate that bias weight in order to investigate its influence on the weight space.

Another way to use the methods proposed in this chapter is to investigate the way learning algorithms influence the way the *network* weight space is searched for a satisfactory solution. Here we have no problems with the various hidden layers since **all** network weights are used. An experimental setup is to train several networks with several learning algorithms. The weights obtained using the various learning procedures and initialisations can be used to train a single SOM (and not one per learning algorithm). It can be expected that for instance the backpropagation rule will move the weights much slower through the hidden unit space than a second order algorithm like the Levenberg-Marquardt method. A problem, however, in the other experiments might be that the weight spaces are very large (on the order of hundreds of weights), the mapping onto two dimensions will result in too much loss of the original space, and it will take too long to train.

Furthermore, it is interesting that redundancy indeed helps the network towards a faster convergence. For instance for the FB it was discovered that, with the average (10 repetitions), the network with 6 hidden units converges 4 times faster than a better network consisting of 2 hidden units. It may be concluded that redundancy helps a network in faster training. A danger, however, is the possibility of a large network being overtrained and thereby cancelling the benefit of faster training. A solution to this problem can be monitoring the nonlinearity. Another problem is that increasing the number of hidden units tends to increase the sensitivity of a network for initialisation. More hidden units make it difficult for a network to get started.

Chapter 5

Classification reliability

5.1 Introduction

The previous chapters were concerned with the monitoring of the generalisation behaviour of a classifier. An alternative approach to check whether or not a classifier has found a satisfactory solution, is to look at the decisions made by the classifier. In practical situations one might have a given classifier without any information available how this classifier was obtained. Therefore, samples classified by the classifier need a reliability measure that states how certain one can be about the correct decision of the classifier. If a poorly trained classifier is used, the reliability of a classification of a sample is expected to be low, whereas an optimally trained classifier is expected to classify a sample more reliably. This chapter will introduce a method for estimating the reliability of a classification of a sample. To be specific, methods for calculating the reliability of classifications made by a feed forward neural network are studied and analysed. Throughout the chapter the reliability measure will be referred to as *confidence* or *confidence value*, since it states how confident one can be in the correct classification of a sample under investigation.

The confidence measure or value can be used in various ways. A first application is the improvement of the classifier's performance using rejection. By using a rejection rule, a sample is returned to an operator for a second opinion. The confidences can be used to reject samples which have a low reliability. This way the confidence can be used to improve the quality of the classifier. Another application is the construction of multi-stage classifiers [Has94, HHS94, KHD96]. Suppose there exists a collection of classifiers which can be used to classify a sample. For each of the samples their confidences can be estimated or calculated. These confidences can be used to create a new classifier that uses them to classify a sample more reliably than any of the original classifiers. In the existing literature it has been shown that classifier combination may be beneficial [TDB97, vBDT97].

This chapter is divided as follows. First the concept of confidence values is introduced. What exactly is meant by a confidence value and how can it be computed? The following section (5.3) deals with different methods to estimate the confidence. Four such methods will be introduced. These estimators need to be tested using an independent test set. This constitutes a problem since independent test sets are hard to get or not available at all. If they are available one needs two of these sets. One for testing the classifier and one for selecting the estimator. Section 5.4 introduces a method to generate validation sets which share the same characteristics as independent test sets. These sets can be used in confidence estimator selection. Finally the chapter ends with a description of experimental results obtained using the confidence and a discussion. Parts of this chapter have been published earlier in [HTD96] and [HDK97] and is largely based on the work by S.A. Tholen [Tho95].

5.2 Confidence Values

When a trained neural network is used to classify an unknown sample one would like to know what the reliability of the classification is, i.e. how certain can one be that the network classification is correct [DLC91, DH73, Dui93b, Fuk90]? In this section it is shown how the probability that an estimate of the class label, i.e. the classification of the network, is correct can be estimated. This reliability, or confidence, and its corresponding estimators will be introduced.

Suppose that a feed forward neural network finds a discriminant function $\mathcal{S}(\vec{x}) = \hat{\omega}$. For each sample $\vec{x}$ the classification based on $\mathcal{S}$ is either correct or incorrect. Therefore, a correctness function $\mathcal{C}(\vec{x}, \omega)$ is defined:

$$\mathcal{C}(\vec{x}, \omega) = \begin{cases} 1 & \text{if} \quad \mathcal{S}(\vec{x}) = \omega \\ 0 & \text{if} \quad \mathcal{S}(\vec{x}) \neq \omega \end{cases}, \tag{5.1}$$

where ω is the true class label of sample $\vec{x}$. When an unknown sample is classified, the confidence of the estimate of the class label, $\hat{\omega}$, is the probability that the classification is correct. Given a classifier, a feed forward neural network classifier in our case, for an (in general unknown) sample the confidence $q(\vec{x}|\mathcal{S})$ or $q(\vec{x})$ for short, is given by:

$$\begin{aligned} q(\vec{x}) &= \sum_{\omega} P(\mathcal{C}(\vec{x}, \omega) = 1|\vec{x}), \\ &= \sum_{\omega} P(\mathcal{S}(\vec{x}) = \omega|\vec{x}), \\ &= \sum_{\omega} P(\hat{\omega} = \omega|\vec{x}, \mathcal{S}), \\ &= P(\hat{\omega}|\vec{x}, \mathcal{S}). \end{aligned} \tag{5.2}$$

The classification confidence is the a posteriori probability for the assigned class $\hat{\omega}$. If the distribution of the samples $\vec{x}$ is known and $\mathcal{S}$ is given, the confidence can be expressed as:

$$q(\vec{x}) = \frac{P_{\hat{\omega}} P(\vec{x}|\hat{\omega})}{P(\vec{x})}, \tag{5.3}$$

where $P(\vec{x}|\hat{\omega})$ is the class conditional density function for class $\hat{\omega}$ (the class to which the sample is assigned) as a function of the sample $\vec{x}$, and $P_{\hat{\omega}}$ is the a priori class probability. Note that equation 5.3 is only influenced by the class overlap and the training of $\mathcal{S}$. In practice, however, the distributions of the samples are unknown. Consequently, the confidence $q(\vec{x})$ has to be estimated. In section 5.3 several possibilities to estimate $q(\vec{x})$ are introduced.

Assume for the moment that $q(\vec{x})$ can be estimated by a $\hat{q}(\vec{x})$. Furthermore, suppose this $\hat{q}(\vec{x})$ can be computed using several methods which result in different estimates for $q(\vec{x})$. Let us denote these different estimates by $\hat{q}_i(\vec{x})$. Consequently a criterion function is needed to compare them. We define a criterion function J_i as follows:

$$J_i = \frac{1}{N_{\mathcal{T}}} \sum_{\vec{x} \in \mathcal{T}} [C(\vec{x}, \omega) - \hat{q}_i(\vec{x})]^2, \tag{5.4}$$

where $\mathcal{T}$ is an independent test set and $N_{\mathcal{T}}$ the number of samples in that test set. The index i is used to indicate the J for the i^{th} estimator used for $q(\vec{x})$. The lower bound of J_i is usually not zero due to class overlap. However, when the underlying class probabilities are known J_{min} can be determined by substituting equation 5.3 for each sample in equation 5.4. The aim is to find the best estimator for $q(\vec{x})$. This can be done to select the estimator which results in the smallest J.

Using an estimator for the confidence, the best classifier for classifying a sample can be selected if a multi-stage classifier is built [TDB97, vBDT97]. The criterion J could be used to select the classifier that classifies the samples with the highest confidence. This requires a fixed estimator for $q(\vec{x})$, otherwise the different classifiers cannot be compared on their classifications.

5.3 Confidence estimators

Since the distributions of the samples and the a priori class probabilities are unknown, there is a need for estimating the confidence values. Generally two types of estimations are considered:

1. The place in the network where the estimator is calculated.
2. The method used to calculate the estimator.

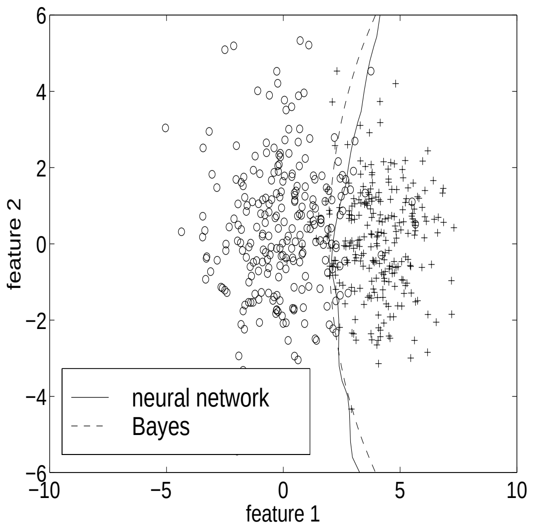


Figure 5.1: *The data set used to train the 2-2-2 neural network. The solid line denotes the Bayes decision function (calculated from the class densities) and the dashed line shows the neural network function.*

5.3.1 Network layer estimation

A feed forward neural network maps the input space onto the hidden layer(s) and then on the output layer. Thereby, $q(\vec{x})$ can be estimated using the methods to be described in section 5.3.2 in the *input* space as well as in the *hidden* unit space or in the *output* space.

Consider the example of two overlapping classes that are separated using a quadratic discriminant function (figure 5.1). Both classes are Gaussian distributed. Class '$\circ$' has a mean $(0, 0)$ and a covariance of 3 in both directions. The other class ('$+$') has a mean of $(4, 0)$ and a covariance of 2 in both directions. A neural network consisting of 2 input units, 2 hidden units and 2 output units was trained on the data set as depicted in figure 5.1. Figure 5.1 also shows the Bayes decision function in the input space. However, if one would consider the hidden unit space, shown in figure 5.2 (left), the classes might be more easily separated and it may therefore be advantageous to estimate $q(\vec{x})$ in that part of the network. As a result the amount of overtraining of the network might determine the best layer to estimate in. For sufficiently trained networks it may be better to estimate $q(\vec{x})$ in another layer than the input space. Note that due to the fact that neural networks make a continuous mapping from on space to the other, probabilities can be estimated in one of the spaces.

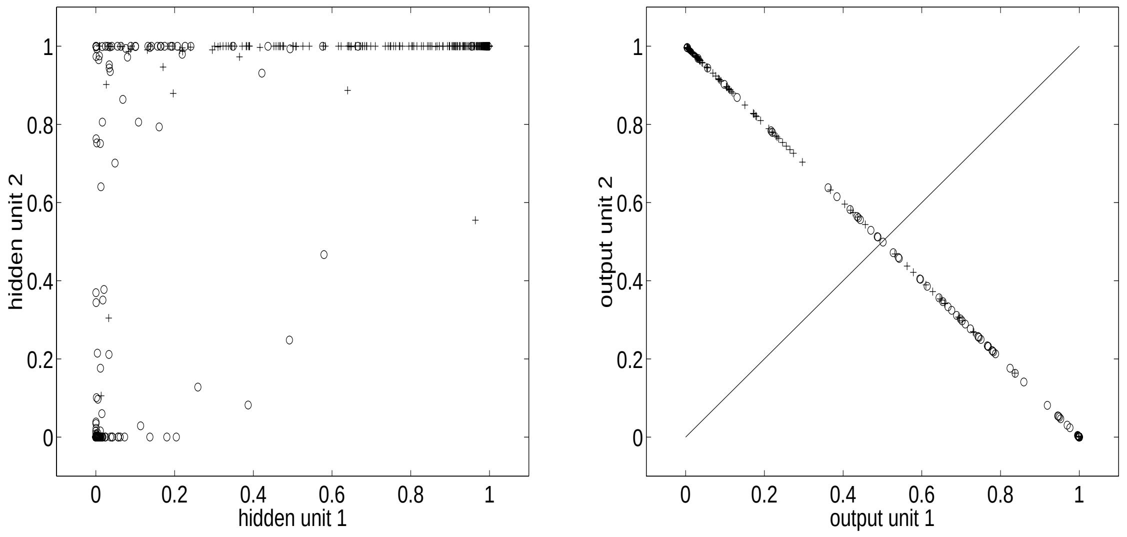


Figure 5.2: *Example of the mappings made by the feed forward neural network. The picture on the left shows the hidden unit space. The figure on the right shows the output unit space with a threshold function at 0.5.*

5.3.2 The estimation methods

There are different methods for estimating the confidence. Here four methods will be introduced that were used in experiments:

1. Network output.
2. *k* Nearest neighbour (*k*-NN) method.
3. Parzen method.
4. Logistic method [And82].

Each of the methods has its advantages and disadvantages. In the experiments we restricted ourselves to methods 1, 2 and 4. Method 3 has been extensively tested [Tho95] but appeared to be outperformed by the other methods. The Parzen method, however, is suited for detecting outliers and is mentioned for the sake of completeness. Now the methods will be discussed in more detail.

Network outputs

This method is the classical way to determine the reliability of a network. The network outputs can be interpreted as the a posteriori probability estimate of the label of a sample. This leads to the following estimator for the confidence:

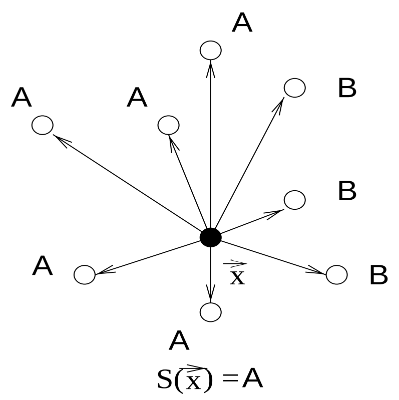


Figure 5.3: *Illustration of the k-NN approach to determine $q(\vec{x})$. The sample under investigation is denoted by the black dot.*

$$\hat{q}_{netout}(\vec{x}) = \frac{o_i(\vec{x})}{\sum_{i=1}^{N} o_i(\vec{x})}, \tag{5.5}$$

where $o_i(\vec{x})$ is the output of unit i representing class ω_i, assuming one output unit per class. The sum is used to normalise the confidence value and N denotes the number of classes (or output units). In general it is not known whether the network is adequately trained. This means that the a posteriori probabilities of the samples may not be estimated accurately enough. Moreover, neural network outputs are in general not directly interpretable as probabilities. This implies either post-processing the network outputs such that they constitute probabilities or adapting the learning algorithm (see for instance [DLC91]) or the error criterion [HIVK93]. The last thing we want to do is adapt the learning rule since it is assumed the networks are given and the way they are trained is unknown. In our experiments we take the network output into account since it is a well established method. Note, however, that this method cannot be used in the hidden or input layer of the neural network. It is not possible to assign a class to a hidden unit (or input unit) as is done for the output units.

k Nearest neighbour method

Given a certain space with a defined distance measure, e.g. the Euclidean distance, the *k*-nearest neighbour density estimator (*k*-NN) can be used to estimate the confidence of the sample under consideration:

$$\hat{q}_{knn}(\vec{x}) = \frac{N_{\mathcal{S}(\vec{x})=\hat{\omega}}}{k}, \tag{5.6}$$

where $N_{\mathcal{S}(\vec{x})=\hat{\omega}}$ is the number of samples among the *k* nearest neighbours sharing the same label as the sample under consideration. The labels of the neighbouring samples are **known** (e.g. from a training set), whereas the label $\vec{x}$ is assigned by the classifier $\mathcal{S}$. Moreover, the more samples "agree" about the label of $\vec{x}$ the more certain we are that the label assignment is correct. This behaviour is illustrated in figure 5.3. Suppose we want to know the confidence of the label assignment of the black dot. Assume furthermore that the surrounding samples have been assigned a class label $\mathcal{A}$ or $\mathcal{B}$. Using the 8-NN approach, the confidence one should have in the correct classification is 5/8, since 5 of the 8 neighbours have class label $\mathcal{A}$. The problem of choosing k still remains, but a general rule of thumb is to use $k = \sqrt{N_{\mathcal{L}}}$ [Das91], where $N_{\mathcal{L}}$ is the number of learning samples.

A disadvantage of this method is that it is computationally intensive for large data sets.

Parzen method

In the Parzen method first the density functions for the classes are estimated and those estimates are used to estimate $q(\vec{x})$. We use a different smoothing factor h_ω for each class ω. It is assumed that a fixed Gaussian kernel is used:

$$\hat{P}_\omega(\vec{x}) = \frac{1}{N_{\mathcal{L}_\omega}} \sum_{\vec{x}' \in \mathcal{L}_\omega} (2\pi h^2)^{-d/2} \exp\left(-\frac{||\vec{x} - \vec{x}'||^2}{2h_\omega^2}\right), \tag{5.7}$$

where $\mathcal{L}_\omega$ is the set of samples having the same label ω. The dimension of the space in which $q(\vec{x})$ is estimated is denoted as d. The smoothing factor h_ω can be determined using a maximum likelihood estimation over the class ω learning set. Having found the densities for each class, $q(\vec{x})$ can be approximated by:

$$\hat{q}_{parzen}(\vec{x}) = \frac{P_{\hat{\omega}} \hat{P}_{\hat{\omega}}(\vec{x})}{\hat{P}(\vec{x})}, \tag{5.8}$$

where $\hat{f}(\vec{x})$ is the estimated joint density function $\hat{f}(\vec{x}) = \sum_\omega P_\omega \hat{f}_\omega(\vec{x})$. Experiments show that the Parzen estimator never outperforms any of the other methods [DLC91, Tho95]. Therefore, the Parzen estimator was not used in the experiments presented in section 5.5.

Logistic method

The logistic method [And82] was chosen for its simplicity. It is a fast method since the parameters need to be calculated only once. The logistic method models the conditional class probabilities using a sigmoid function. For a two class problem the probability that $\vec{x}$ belongs to the correct class is given by:

$$\hat{q}_{logistic}(\vec{x}) = P(\mathcal{C}(\vec{x}, \omega) = 1 | \vec{x}) = \frac{1}{1 + \exp(\vec{\beta}\vec{x} + \beta_0)}, \tag{5.9}$$

assuming equal prior class probabilities. The parameters to be determined are $\vec{\beta}$ and β_0. The total number of parameters is given by the dimensionality of the space $\vec{\beta}$ plus one for a bias β_0. They can be estimated using a learning set and an optimisation technique like Newton's method. As stated earlier, equation 5.9 only holds for the two class case, however it can be extended to more classes quite easily. This is done by taking one class as a base class, called g, and estimating all the parameters of the other classes with respect to that base class. Rewriting equation 5.9 for the base class approach results in:

$$\hat{q}_{logistic}(\vec{x}) = \left(1 + \sum_{i=1}^{N_g-1} \exp(-\vec{\alpha_{gi}}\vec{x}')\right)^{-1}, \tag{5.10}$$

where $\vec{x}' = (\vec{x}, 1)$ and α_{gi} is the $(\vec{\beta}, \beta_0)$ for class i with respect to the base class g and N_g denotes the number of classes present.

5.4 Validation sets

In the previous section, four estimation methods for $q(\vec{x})$ have been introduced. The best one should be found using an independent test set (see equation 5.4), as use of the learning set generally causes biased results. In order to avoid splitting of the learning set into a smaller one and a test set, we studied the possibility of generating an artificial validation set. Such a validation set should roughly have the same characteristics as the learning set but should be distinctive enough to act as an independent test set.

To create a validation set, the *k*-NN data generation method [Dui93a] was used. The learning set is used as a starting point for creating a new set. Around each sample in the learning set, new samples are generated using Gaussian noise around the original sample. The samples are generated in such a way that the local probability density is taken into account. This local probability density is estimated using the *k* nearest neighbours around an original sample. The estimated probability density is used as the probability density function to generate a new sample from the existing samples. In the experiments *k* was chosen to be the (dimensionality of the space)+2. Other experiments ([SRD98]) show that it is not beneificial to take a large *k*. Figure 5.4 shows the effect of the *k*-NN data generation method using a learning set. The original set contained 50 samples, using additive random variates Gaussian distributed around $(0, 0)$ an, the new set was generated using 5 neighbours and consists of 150 samples. The effect of the local density estimation can be observed, since the new set follows the shape of the original set. Also the sensitivity to outliers is shown. Consider again figure 5.4, the right picture. It shows two outlying samples pointed to by the arrows. These are due to the fact that the original set has an outlier, a 5-NN approach was used and more samples were generated than available in the original set.

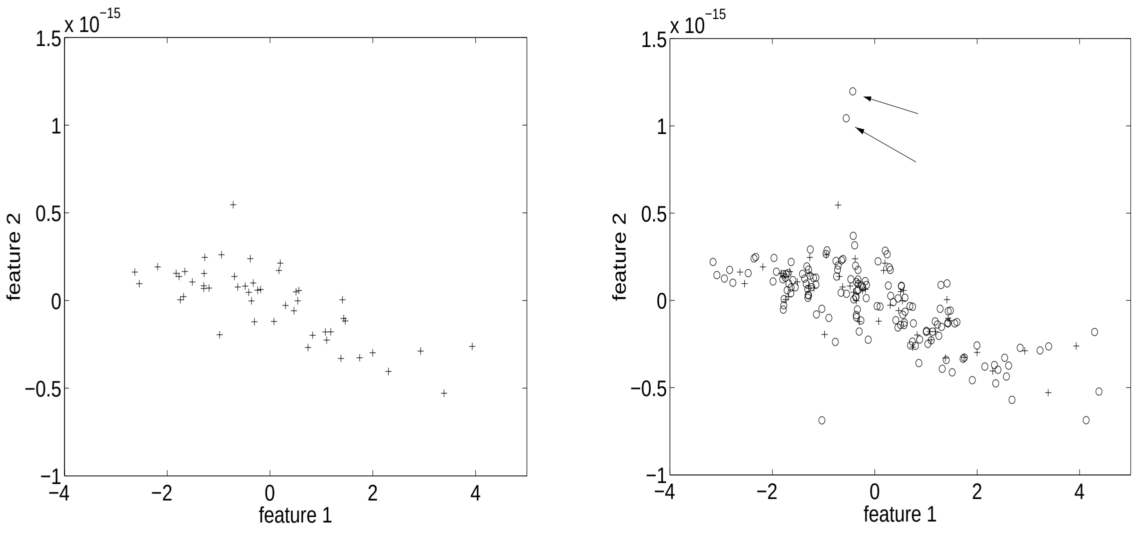


Figure 5.4: *The effect of the k-NN data generation method. On the left the original set is shown, on the right the generated set. The arrows point to the outliers generated using the k-NN generation method.*

This lead to the replication of samples and consequently some outliers.

Note that a validation set will **not** be able to replace an independent test set entirely, but it is an acceptable alternative. Experiments show that the *k*-NN data generation method generates a validation set which is reliable enough to select the same estimators (e.g. logistic) and the network layer (e.g. output layer) as an independent test set would do.

More experiments using a validation set can be found for instance in [Tho95]. These experiments show the validity of the data generation method for poorly trained neural networks. Also the validity of the data generation method is analysed in section 5.5 where results of experiments are shown.

5.5 Experiments

In order to test the applicability of the confidence value estimation in classifier analysis, several experiments were performed. Here three experiments are described that use artificially constructed data sets. The first experiment shows that the place to estimate the $q(\vec{x})$ depends on the amount of overtraining of the neural network. In the second experiment the usefulness of the data generation method is shown on the 'Overlap' data set (see chapter 3 figure 3.2). This set was also used in the third experiment which shows what happens if the confidences are used as rejection criterion.

The first experiment uses only the k-NN method and the network output for estimating the confidence, but estimates the confidence in different parts of the network. In the second and third experiment the logistic method was included. A practical application of the confidence values can be found in [Tho95, HTD96] and in chapter 6.

5.5.1 Network layer selection

The experiments with this artificial data set (figure 5.5) were done to show that the amount of overtraining influences the place in the network to estimate the confidence. A 2-3-2 network architecture (2 inputs, 3 hidden units and 2 outputs) was trained on the data set. The network was trained using fast backpropagation (FB) as explained in chapter 4. Three different sets were generated: a training set consisting of 1,000 samples per class, an independent test set consisting of 1,000 samples and a validation set of 500 samples per class. This validation set was used in the training of the network to find the best network. By using a validation set, a decision function can be found without using the independent test set. The sets could be generated since the distributions of the samples are known a priori.

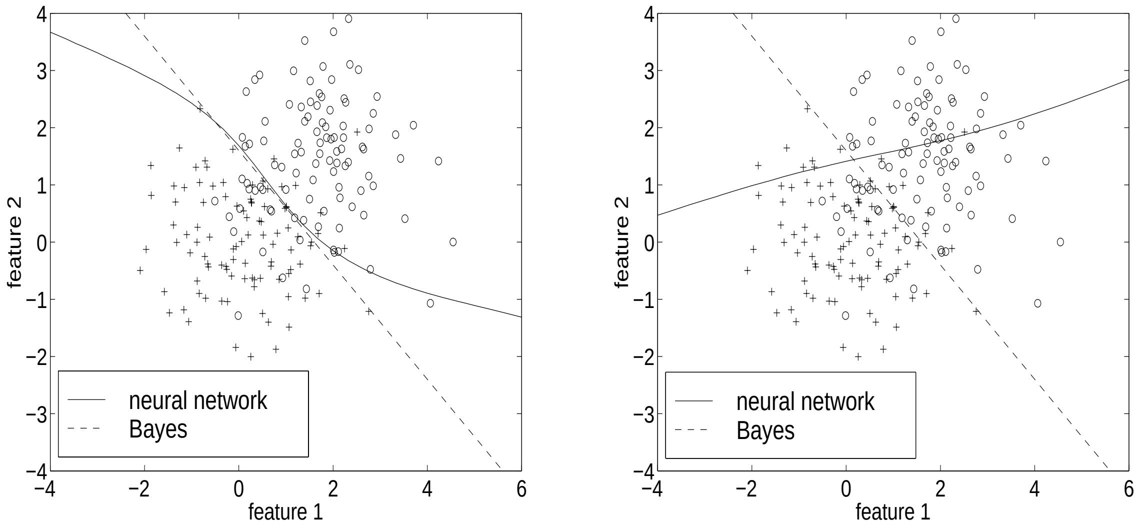


Figure 5.5: *The left picture shows the training set and corresponding best decision function found after training. The picture on the right shows the decision function of the poorly trained network.*

The two classes have about 10% overlap (see figure 5.5), equal covariance matrices (being the unity matrix) and different means, $(0, 0)$ for class $\mathcal{A}$ and $(1.6, 1.6)$ for class $\mathcal{B}$ the Bayes error is 0.13). In figure 5.5 (left picture) the training set is shown with the best decision function (it has an error which is closest to the Bayes error) found by the FB network training method based on the validation set. Also a poorly trained network was generated by performing only a single update step after initialisation. In the right picture of figure 5.5 the poorly trained network is shown. Consequently we have two networks, one well trained network referred to as "good" and one poorly trained network referred to as "bad". Using these two networks the influence of the training of the network on the estimators is investigated. This is done for the *k*-NN method, for several *k* values ranging from 1 to 1,000 and for the q_{netout} estimator.

The J for the different layers of the network where the estimator was used, was determined using an independent test set of 1,000 samples per class. This test set was also used to calculate the network performance and to estimate the confidences. Since the distribution of the samples is known, J_{min} can be estimated. The results of the *k*-NN estimator can be compared to this value. Table 5.1 summarises the results for the 2-dimensional experiment. Between parentheses the standard deviation is shown. The *k* was chosen to be $\sqrt(2000) \approx 45$.

Table 5.1: *The estimation results for the k-NN method for the "bad" and the "good" network on the two class 2-dimensional data set. Between parentheses the standard deviation is shown.*

estimator	J "good"	J "bad"
***k*-NN**		
input layer	0.0923 (0.004)	0.0923 (0.004)
hidden layer	0.0947 (0.004)	0.1030 (0.004)
output layer	0.0947 (0.005)	0.1848 (0.005)
netout	0.0986 (0.004)	0.2140 (0.002)
J_{min}	0.0800	0.0800
performance	86.95%	68.30%

For the "good" network it shows that the best layer to estimate $q(\vec{x})$ is the input layer, i.e. the input space. Due to the fact that the network is trained well, it approximates the a posteriori probabilities best in the input space. This is also shown by the netout values. These are the actual outputs of the network used as the estimator for $q(\vec{x})$. The differences between the layers, however, are small given the standard deviations.

For a poorly trained network the selection is clear. The input estimator is the best one compared to the others, even when taking the standard deviation into account. Using the network outputs in this case is certainly a bad decision. Due to the fact that it is poorly trained the a posteriori probabilities are not approximated well and therefore q_{netout} is unreliable.

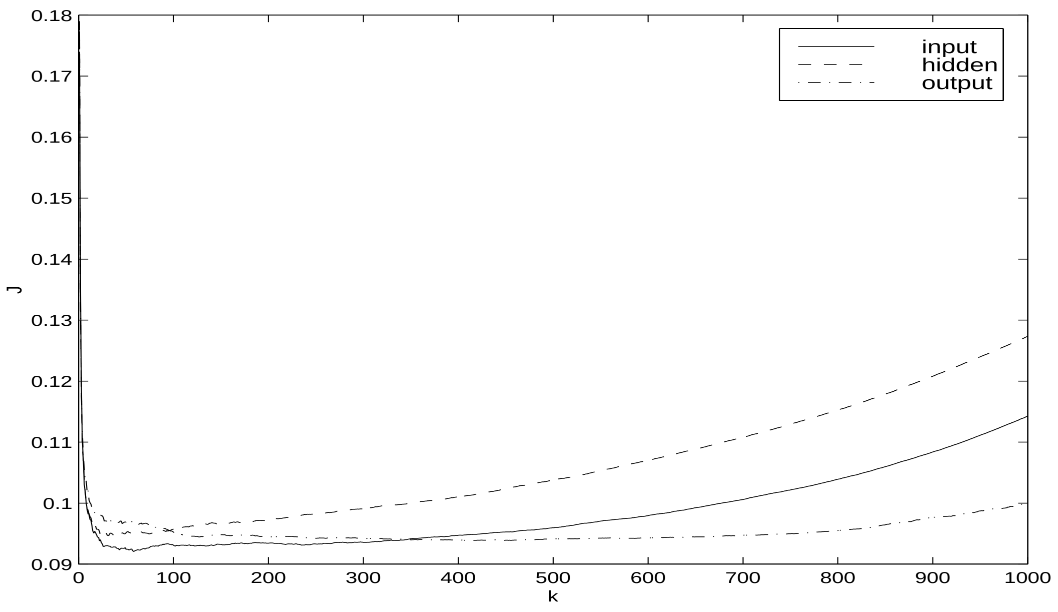


Figure 5.6: *In this figure the Js of the k-NN estimator for various k are drawn. The Js were computed for the “good” network.*

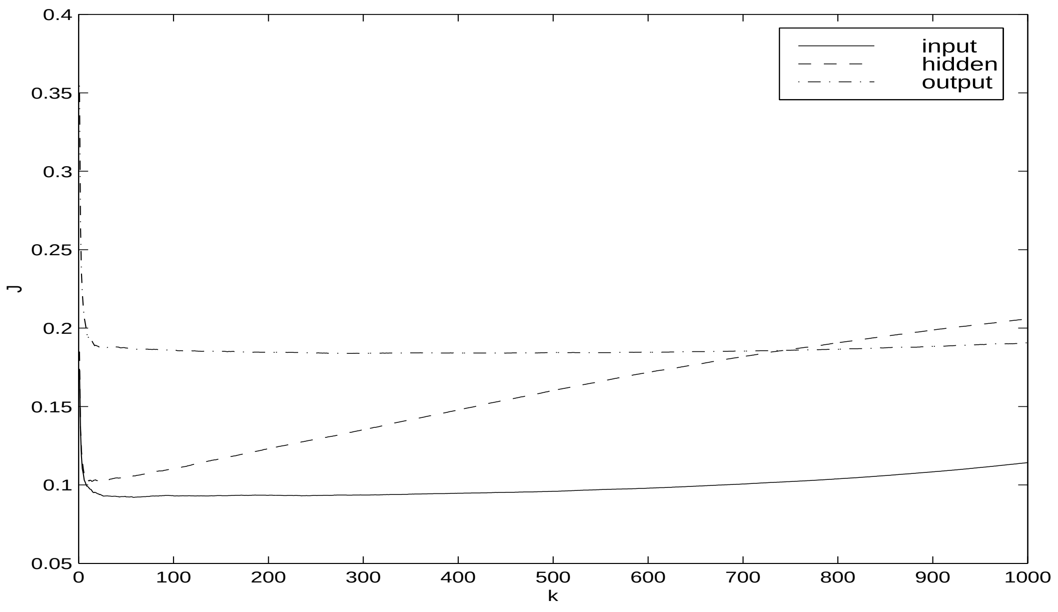


Figure 5.7: *In this figure the Js of the k-NN estimator for various k are drawn. The Js were computed for the “bad” network.*

The k, however, was fixed for the estimators in the different layers. In each layer a k of $\sqrt{2000} \approx 45$ was used. An option would be to vary k in the layers. In figure 5.6 and 5.7 the different J for various k have been depicted per layer. The value of k ranged from 1 to 1,000, i.e. 50% of the total data size. For the "bad" network (figure 5.7) it shows that the confidence estimator in the input is always the best, whereas for the "good" network the estimator in the output space becomes better for k larger than 400. Moreover, the differences between the Js in the various layers is small indicating that the network is not in an overtrained situation. If the network would be poorly trained, there should be a clear choice for the input layer as shown in the experiment.

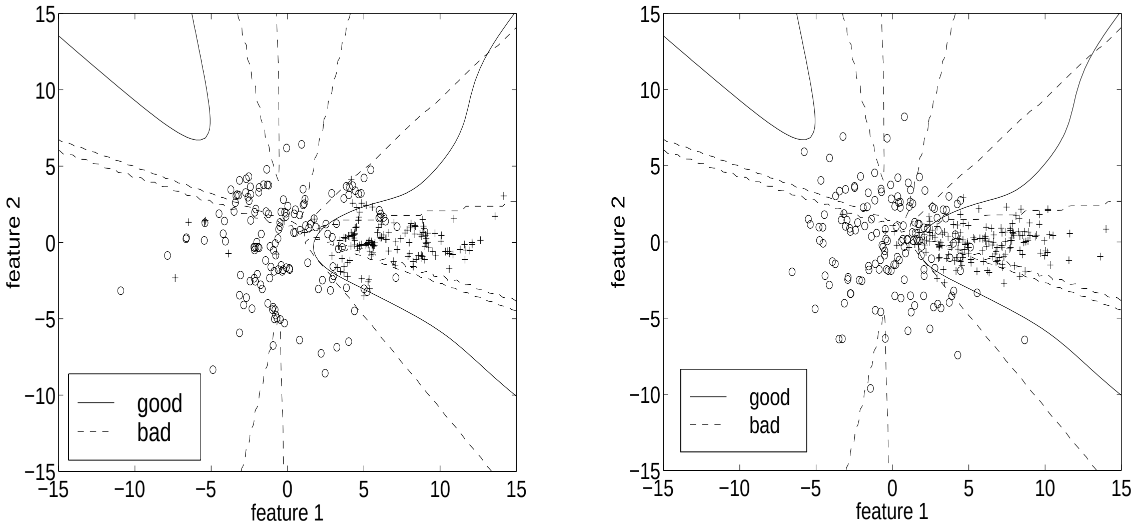


Figure 5.8: *The picture on the left shows the validation set generated using a 2-NN data generation. The right picture shows an independent test set of the same size. In both pictures the decision functions of a "good" and a "bad" network are drawn.*

5.5.2 Validation set generation

The following experiments were conducted to investigate whether the k-NN data generation method is a valid method to act as a replacement for an independent test set. It is expected that computing J for both the test set, which can be generated since the distributions are known, and the generated set have almost equal performance. That is, for both sets the best layer and confidence estimator should be the same. The data generation method, however, remains sensitive to outliers. The 'Overlap' set was therefore used since it contains some outlying samples and is relatively small. This enables the analysis of the behaviour of the data generation method on outliers. In figure 5.8 (left) the k-NN generated validation set is

Table 5.2: *The table shows the results for a validation set and an independent test set in the layers of the network and various confidence estimators. The results are derived using a well trained network (see figure 5.8).*

estimator	$J_{validation}$	J_{test}
***k*-NN**		
input	0.0838	0.0793
hidden	0.0832	0.0750
output	0.0777	0.0792
logistic		
input	0.1093	0.1030
hidden	0.0830	0.0880
output	0.0774	0.0694
netout	0.0775	0.0809
performance	89.67%	91.67%

depicted. It was generated using $k = 3$ (the dimensionality of the inputs space + 1) and from the original 80 samples in the learning set, 300 new ones were generated, i.e. 150 samples per class. Also a test set containing 300 samples was generated (on the right in figure 5.8).

The difference between the two data sets can clearly be observed in figure 5.8. The validation set is influenced by the outliers present in the underlying training set (chapter 3 figure 3.2). The main question is therefore how this influences the behaviour of the estimators. A neural network using the FB method was trained on the 'Overlap' set. After training the J for the *k*-NN estimator and logistic estimator were calculated. The logistic estimator was used for the sake of completeness. It can be expected that due to the fact that a *k*-NN method was used, the *k*-NN estimator might be favoured over any other estimator. Table 5.2 summarises the results for the estimators on both sets.

As can be observed from table 5.2 the network has a small performance loss when using the validation set. This was to be expected since the set has "aligned" to the outliers present in 'Overlap'. The loss, however, is not such that the best estimators become significantly different. For both sets it holds that for the logistic estimation, the output layer is the best. Furthermore, in both cases, the q_{netout} estimator indicates that the network is not in an overtrained situation. The results for the *k*-NN estimator are comparable to the logistic estimator. Also in this case for both sets the confidence can be best estimated in the output layer of the network. There is no sign of bias towards the *k*-NN estimator as was feared due to the nature of the generated set. The fact that the J in the input layer of the network is larger for

Table 5.3: *The table shows the results for a validation set and an independent test set in the layers of the network and various estimators on an overtrained network (see figure 5.8).*

estimator	$J_{validation}$	J_{test}
***k*-NN**		
input	0.0838	0.0838
hidden	0.0910	0.1074
output	0.1059	0.1414
logistic		
input	0.1093	0.1029
hidden	0.1600	0.1647
output	0.1078	0.1443
netout	0.1068	0.1353
performance	86.67%	82.33%

the logistic method than for the *k*-NN method is because of the fact that the logistic function is a linear estimator whereas the *9*-NN ($\sqrt{(}80)$) estimator is nonlinear. The best separation function for the data set is a quadratic separator, which is nonlinear. This function should give the best estimations if it would be implemented by a neural network and used the q_{netout} estimator. In general it can be concluded that the *k*-NN data generation method is a valid method to use when no independent test set is available.

5.5.3 Rejection

The last experiment involved the use of the confidence values as rejection criterion. As stated in the introduction, samples with a low confidence can be rejected in order to improve the quality of the classifier. Hopefully, the number of rejected samples is low compared to the total number of samples. If not, it may be the case that the network is in an overtrained situation. In this experiment the data from the previous subsection was used to inspect the rejection behaviour. Since the previous experiment did not use an overtrained classifier, a new overtrained classifier was generated. This also requires the computation of the confidences and corresponding Js. Table 5.3 summarises the results for that network.

Remarkable, when observing table 5.3, is the fact that the generated data set is better classified than the independent test set. This is probably due to the shape of the classification function which is better suited for the generated set. It turns out that at the overlapping part of the classes, the test set has more samples (see figure 5.8). This is to be expected since the test set better represents the class densities. The interesting thing is to observe the rejection

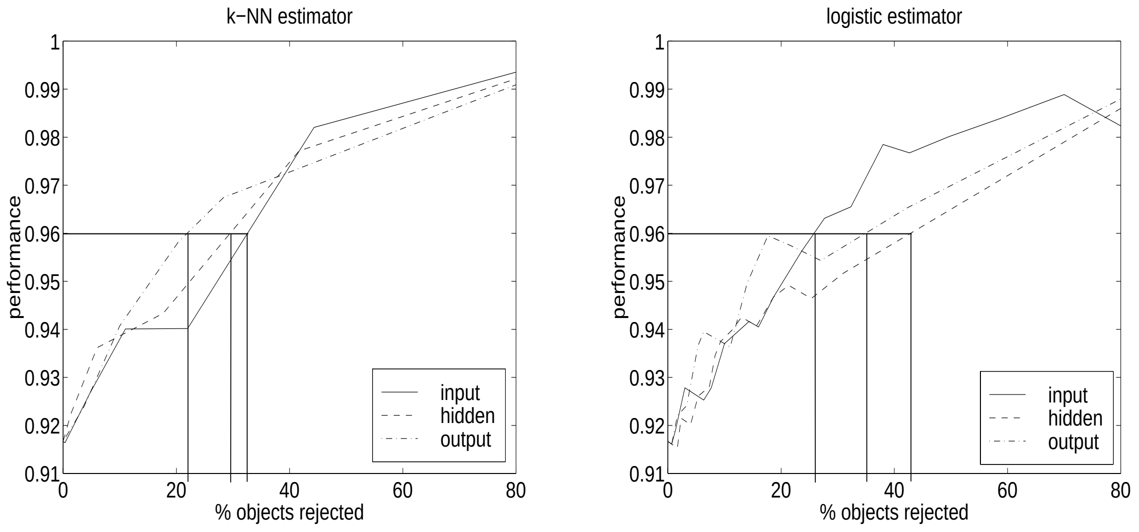


Figure 5.9: *The figures show the rejection curves for two estimators in the different layers of the neural network. On the left the k-NN estimator is shown and on the left the logistic estimator. The lines drawn in the figures show the % of objects that should be rejected to obtain a performance of 96%.*

behaviour. It is expected that when rejecting samples the classification performance will increase. Although it may be the case that false rejections are done. On the average, however, it is expected that the "correct " samples are rejected. The most obvious thing to do is to reject samples using the best estimator. For the test set this implies that both in the case of the "good" network and the "bad" network the logistic estimator in the output space should be used. In case of the generated data set it should be the logistic estimator in the output space, and the *k*-NN estimator in the output space.

Figures 5.9 and 5.10 show what the classification performance of the network is for an estimator when rejecting a percentage of the samples. The performance was calculated using a threshold on the confidences. Samples are rejected when they do not meet the threshold, also the number of rejected samples was stored. The curves show on the average an increase in performance when classifying the remaining samples. The most interesting property of the curves is their slope. This slope determines the expected improvement in performance when rejecting samples. The steeper this slope is, the less samples need to be rejected to gain significant performance increase. Furthermore it is expected that the best estimators show the steepest slopes. Consider again figure 5.9, here the rejection curves are drawn for the logistic estimator and the *k*-NN estimators in the different layers of the network. From table 5.2 it is expected that the estimators in the output space show the steepest slope when rejecting only a few samples. When rejecting only a few samples the performance of the network

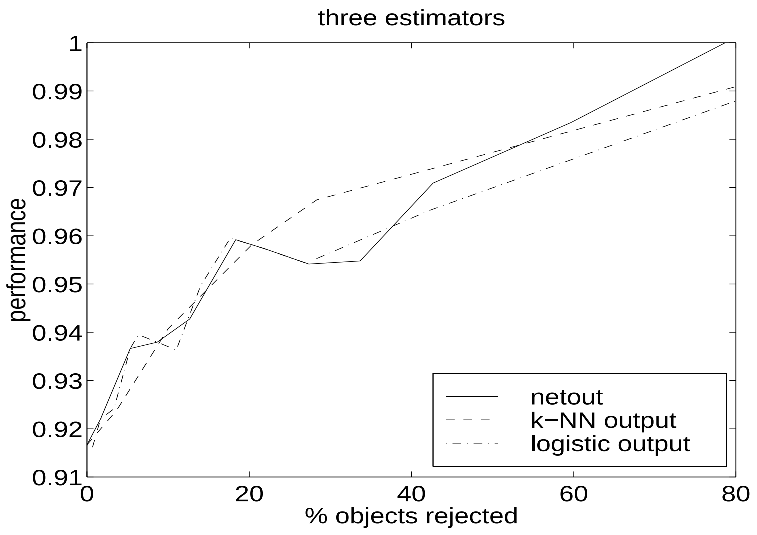


Figure 5.10: *This figure shows the rejection curve for the two best estimators of the k-NN and logistic method plus the rejection curve when using the network outputs.*

increases the most for the best estimators, although for the *k*-NN method the estimator in the hidden unit space seems to be the best for small rejection rates. In figure 5.9 the percentage of objects is marked to obtain a 96% performance. It shows that for *k*-NN estimators that rejecting about 22%, 30% and 35% of the objects results in 96% performance. In the case of the logistic estimator, a larger number of objects need to be rejected (about 25%, 35% and 43%). When using the estimators as rejection criterion, one should use the *k*-NN estimator in the input space.

In figure 5.10 the rejection curves for the best estimators, logistic in the output space and *k*-NN in the output space and the netout are drawn. Both the logistic and *k*-NN estimator perform equally well, which was expected from the results in table 5.2. Also the network output seems to be a good estimator, although it shows a decrease when rejecting about 40% of the samples, but then it “catches up”. This is probably due to the fact that at approximately 40% the best separating function in the output space is found. The causes on the average more correctly classified samples to be rejected.

5.6 Discussion

In this chapter definition of confidence values (equation 5.2) was introduced to judge the reliability of a classifier. By calculating such a reliability of the classification of samples a decision can be made whether one should select that classifier or not. Moreover, the confidences can be used to create multi-stage classifiers, although this is not shown here. Current research [TDB97, vBDT97] shows how such values can be used for creating such classifiers.

Furthermore, it has been indicated how the need for an independent test set can be circumvented using the data generation method which creates a data set from an existing training set. A problem with the method, however, is that it is not an entirely independent set. It has the same characteristics as the set from which it is generated. This may cause problems if it has outliers. These outliers may cause new samples in the validation set which may not be representative for the test set. In our experiments, however, this method results in the same predictions as an independent test set. Although for poorly trained neural networks its performance might be better than expected due to outlier sensitivity.

Furthermore several methods were introduced to estimate the confidence of a classification. The well known network output method was included for completeness. Using this method usually leads to adapting the learning rule or normalising the network outputs. The other methods differ in their complexity of application. The most convenient one is the logistic estimator, this method requires the parameters just once. It requires, however, some adaptation when applying it to multi-class cases. In multi-class cases the *k*-NN method is easier to apply. It will also directly give the confidence for a sample per class. The Parzen method is the most complex one used in confidence estimation. In high dimensional feature spaces this method breaks down [DLC91, Tho95].

Experiments show that the confidences indeed predict whether the classifier is poorly trained. However, one is not able to state whether it is undertrained or overtrained. In order to be able to conclude whether a network is adequately trained one needs a tool like the nonlinearity which was presented in chapter. The confidence values seem to be especially useful as rejection criteria. By rejecting those samples which have a low confidence the performance of a classifier can be significantly increased. It was shown that the best confidence estimators the best are also the ones which cause the largest increase in network performance when rejecting only a few samples. A problem, however, is “false rejection”. Based on the confidences one might reject samples which were classified correctly. This is inherent to rejection.

Chapter 6

Digit Recognition

6.1 Introduction

In the previous chapters, artificial data sets were used to show the validity of the introduced methods and techniques. The advantage of such data sets is that their properties are known in advance. This enables an accurate and precise analysis. The methods, however, are intended for use in situations where one has no such prior knowledge. Therefore, we will focus on a real-life problem in this chapter: the classification of *handwritten digits*. Neural networks have been shown to be successful in this area, see for instance [LCBD$^+$89, LCS91] and [dR96]. Our aim, however, is not to find an optimally performing neural network (which is usually the target) but merely to demonstrate the methods of chapter 3 and chapter 5. Due to the dimensionality of the data set, the symmetry breaking visualisation (chapter 4) could not be performed in a reasonable amount of time and therefore the symmetry breaking behaviour was not studied. The larger the dimensionality of the feature space, the more computer memory is required (due to the nature of the SOMPAK package [KHKL95]). This slows the processing time. For instance, the experiments shown in section 4.4 (chapter 4) take about two weeks whereas an experiment with a 267-dimensional feature space takes 2 months.

In the literature the problem of handwritten character recognition using neural networks has been studied by several people [LCBD$^+$89, LCS91, dR96]. Our approach differs in the use of the architecture and objectives. Commonly used architectures in the literature are the *shared weights* neural networks [LCBD$^+$89]. These networks have a more complicated structure than the standard feed forward neural networks as presented in chapter 1. The advantage, however, of weight sharing is that the number of adjustable parameters is decreased. Furthermore, our objective in these experiments is to show the use of the methods proposed in this thesis. In the literature, however, one is usually interested in finding the "best" performing network.

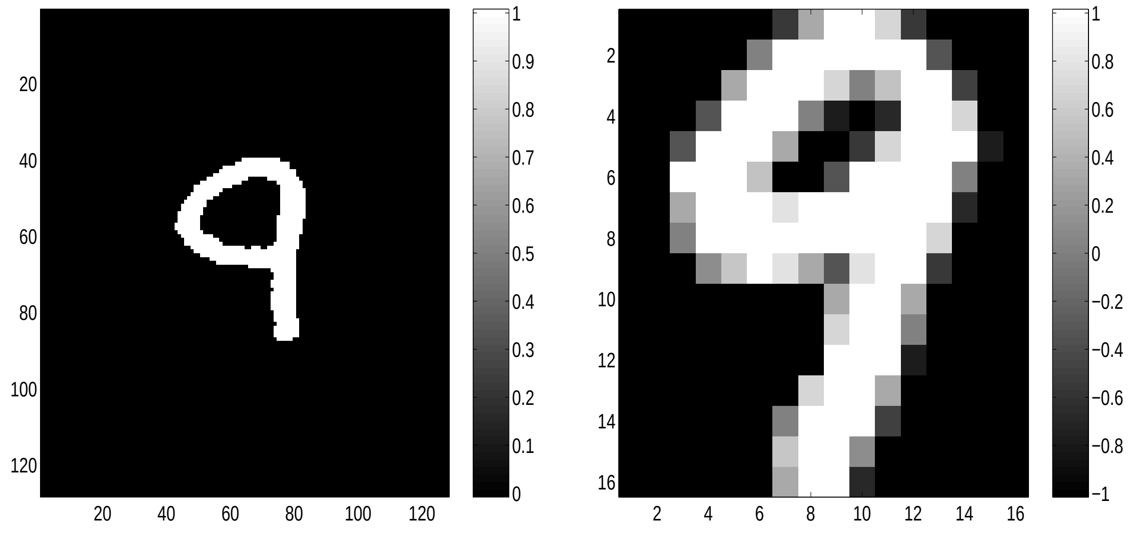


Figure 6.1: *The figure on the left shows the original NIST image (*128×128 *pixels), on the right the pre-processed* 16×16 *pixels image is shown.*

We are interested in showing the validity of our methods in an "unknown" environment.

The chapter is divided as follows. Section 6.2 will introduce the data set and the neural network used. The next section will show the results of the experiments on the data set by performing an error analysis, a nonlinearity analysis and confidence value estimation. The chapter ends with a discussion regarding the results presented. Due to the large amount of available results only a part will be presented in tables and figures.

6.2 Experimental setup

6.2.1 Data set

The data sets used were derived from a subset of the NIST database. The NIST [WG90, WG92] database contains hundreds of scanned handwritten forms. In the database these forms are present at various levels of segmentation. We only used the segmented digits. These digits are binary images of 128×128 pixels, which are too large to train a neural network within a reasonable amount of time. Therefore, they were pre-processed to real-valued images of 16×16 pixels. The pre-processing steps consisted of *shearing*, *line width normalisation* and *scaling*. These steps are described in full detail in De Ridder [dR96]. In figure 6.1 an example of the pre-processing is shown.

The 16×16 images are used to train the neural networks. This means that the neural network is trying to classify points in a 256-dimensional. In order to be able to compare the results with others, for instance [DdR97], three main data sets were created, a training set, a test set and a validation set. The training set was divided into 6 batches consisting of, 5, 10, 25, 50, 100 and 250 samples per class. On each of these batches a neural network was trained. Three of the batches (25, 100 and 250) were used for nonlinearity analysis and confidence value estimation. The test set consisted of 1,000 samples per class and the validation set had 500 samples per class.

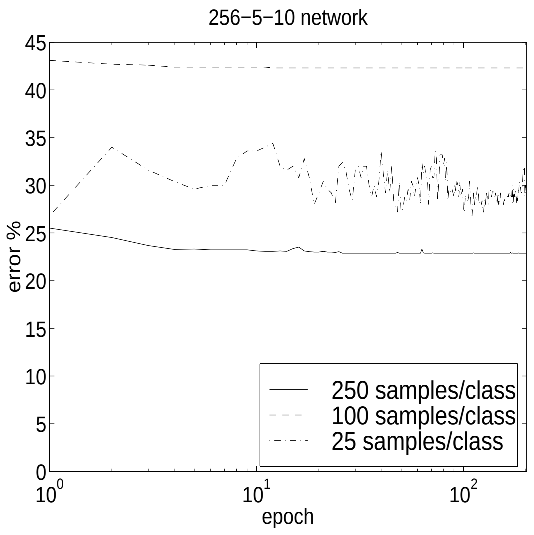
6.2.2 Neural networks

A standard feed forward neural network (see chapter 1) was used to classify the digits. It was trained using the training sets and the backpropagation algorithm present in ANN / SPRLIB [HKdRS96]. The analysis of the results was done in Matlab. The network consisted of 256 inputs, one input for each pixel in the image, a variable number of hidden units and 10 output units, one class per unit. The number of hidden units influences the training results of the network and ranged from 5 to 20 hidden units: 5, 10, 15 and 20. For these numbers networks were trained for a fixed size of the training set. Three networks were trained per size of the hidden layer and one for each size of the training set. Due to computational time limitations, only networks were analysed which were trained using standard backpropagation with momentum.

In each training cycle the same parameters were used. All the networks were trained for a maximum of 5,000 epochs with a momentum term of 0.9 and learning rate of 0.3. An alternative would be to use a validation set for stopping. We are, however, interested in a complete training cycle and not only the one resulting in the best network given the validation set. Therefore, the validation set was included for the sake of completeness. The training was stopped when the maximum number of epochs was reached (5,000) or when the mean squared error dropped below 0.001. The state of a network, i.e. the weights, were written to disk every 25 epochs. This results in 202 network files (maximally), i.e. the first epoch, last epoch+1 (e.g. 5,001) and 5000 / 25 epochs. The individual network files were used for nonlinearity analysis and confidence value estimation.

6.3 Results

The hidden unit space in which the units “move” around is 267-dimensional (256+10+1). Depending on the number of hidden units, the maximum number of vectors used to train is about 2,020, which is not that large. The SOM, however, would be a grid of typically 150×150 units and each unit would have a 267-dimensional prototype vector. In the cur-



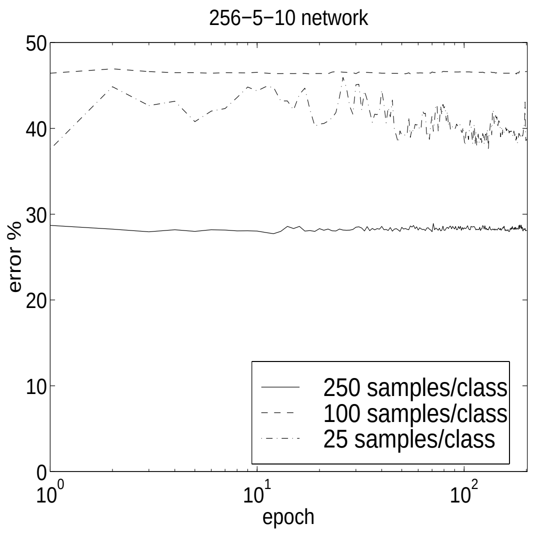


Figure 6.2: *The figure shows the errors for a network consisting of 5 hidden units. On the left the training error is shown and on the right the test set error.*

rent implementation this makes it too slow due to the amount of matrix calculations done. Therefore, no experiments to investigate the symmetry breaking behaviour have been done.

6.3.1 Error rate analysis

The first thing which is important to know is whether the training has converged to a reasonable situation. This can be checked visually by inspecting the curves for the training set error and test set error. In figures 6.2, 6.3, 6.4 and 6.5 (left figures) these curves are shown for 5, 10, 15 and 20 hidden units. Note that not all networks have been trained for the maximum number of epochs. For example, in figure 6.4 the network trained on 25 samples / class already converges in an early stage. Investigation shows that the training process indeed converges to a more or less stable situation. The error level on which this occurs varies from network to network. The 5 hidden unit network (figure 6.2 on the left) is clearly too small to be able to properly classify the training sets. The errors remain almost constant after the initial phase, about 23% for 250 samples / class and about 42% for 25 sample / class. The error for 100 samples / class fluctuates around 30%. Quite a small error is reached for a network of 20 hidden units. Since it has the largest number of parameters and it is expected to have an almost zero error. For the case of 25 samples / class this indeed happens. The error reduces almost to zero, not completely zero due to the fact that the mean squared error (MSE) does not need to be zero.

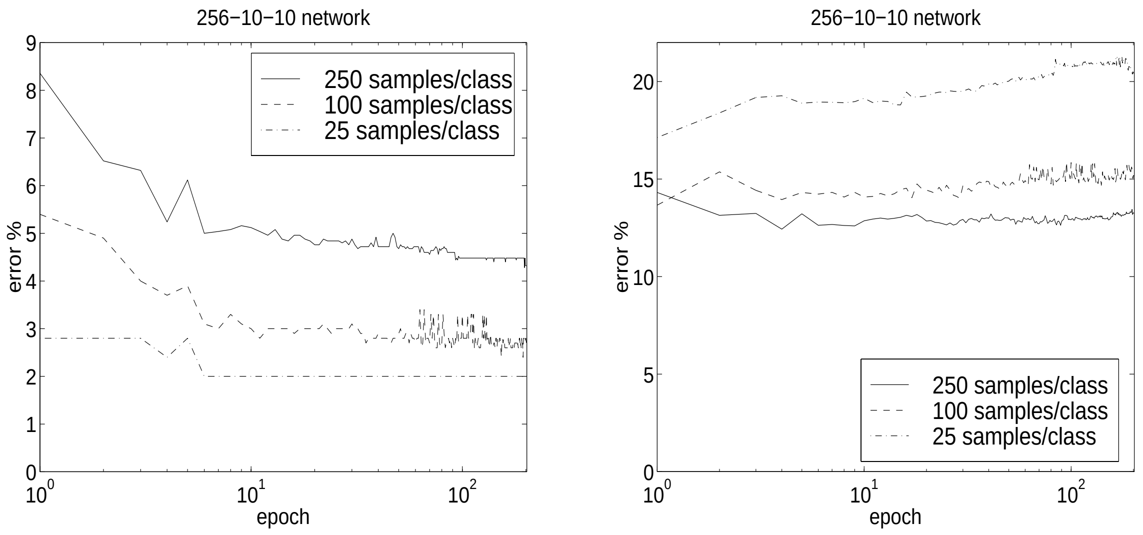


Figure 6.3: *The figure shows the errors for a network consisting of 10 hidden units. On the left the training error is shown and on the right the test set error.*

Judging the network performance only by means of the training set error is not sufficient. By inspection of the test set error, one is able to state whether the networks found after training are not in an overtrained situation. Figures 6.2, 6.3, 6.4 and 6.5 (on the right) show the test set error curves for the networks. The curves can be compared with the learning set error to find a best network (cf. the curve shown in figure 1.5 chapter 1). Note, however, that selecting the best classifier using a test set is in fact a way of training.

Let us consider the right graph in figure 6.5. Here the test set errors for the 20 hidden units network are depicted. As expected the test set error shows an increase towards the end of the training cycle. This behaviour is best shown for the network trained on the 100 samples / class. The error for the 250 samples / class is just slightly increasing. Table 6.1 summarises the training results.

As can be observed in table 6.1 the network with 20 hidden units has the lowest error. This was expected given the data set. How does this compare, however, to other methods and experiments? It turns out that the best network (20 hidden units) has an error which is about twice as high as the error made by a *1*-NN classifier. Moreover, the network is worse than the ones obtained by Le Cun and de Ridder. They obtain results near the 4% error. However, they used the more complex shared weights network. This network has less parameters than a traditional feed forward network, but is more complicated to train.

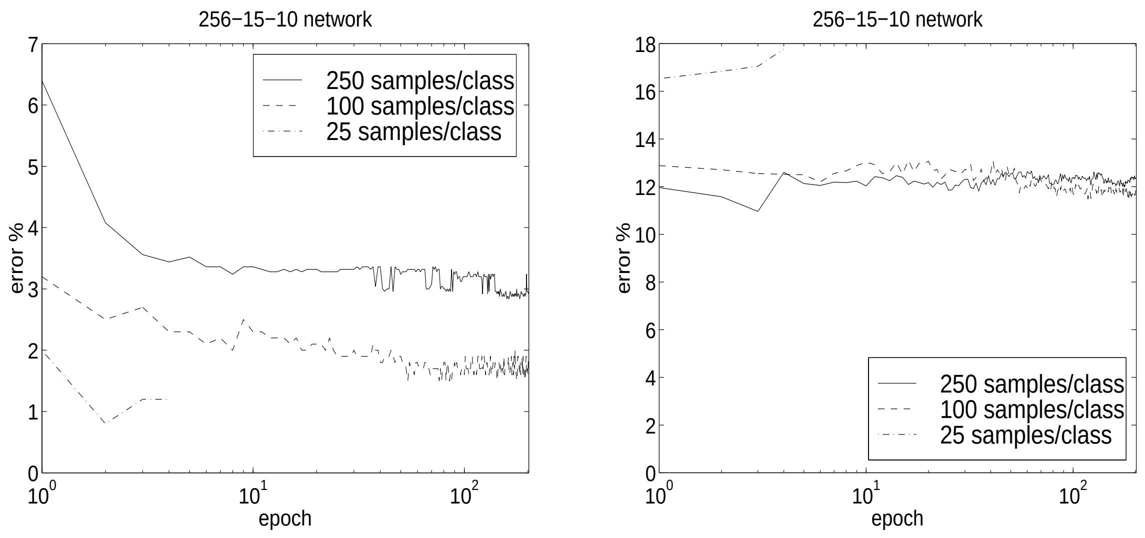


Figure 6.4: *The figure shows the errors for a network consisting of 15 hidden units. On the left the training error is shown and on the right the test set error.*

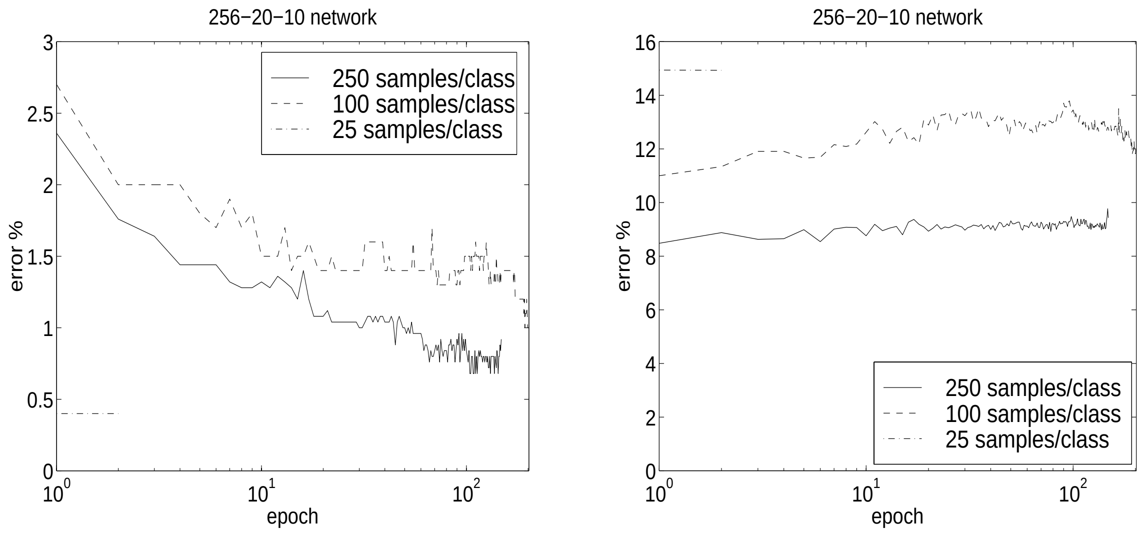


Figure 6.5: *The figure shows the errors for a network consisting of 20 hidden units. On the left the training error is shown and on the right the test set error.*

Table 6.1: *This table summarises the results of the training cycles. It shows the lowest errors obtained during training and testing.*

# hidden units	Training set size					
	25		100		250	
	train	test	train	test	train	test
5	26.80%	37.37%	42.30%	46.33%	22.88%	27.72%
10	2.00%	17.13%	2.40%	13.66%	4.28%	12.43%
15	8.00%	16.53%	1.50%	11.48%	2.84%	10.96%
20	4.00%	14.93%	1.00%	11.00%	0.68%	8.48%

6.3.2 Nonlinearity analysis

In chapter 3 the nonlinearity measure $\mathcal{N}$ was introduced to monitor generalisation behaviour in such a way that overtraining might be detected. For each of the trained networks their nonlinearity during training was estimated using the approximation algorithm (chapter 3, section 3.2.2). The number of samplings used to determine the nonlinearity was set to ten times the number of samples in the data set under investigation. Figures 6.7 and 6.6 show the nonlinearity curves for networks trained on 25, 100 and 250 samples per class and various hidden unit sizes. The curves shown in the figures are polynomial approximations of the real nonlinearity curves. This approximation was done for the sake of clarity. We used a polynomial of degree 10 to approximate the curves and the Matlab [Inc96] *polyfit* and *polyval* functions.

Consider the nonlinearity curves for the best network (right graph in figure 6.6). The curve for the 25 samples/class case is hardly visible since that network converged very quickly. The other two curves are more interesting. Although they fluctuate, their nonlinearity is relatively small (the maximum nonlinearity is about 0.028). The solutions obtained by the networks are close to linear. It is not surprising since the results obtained by our simple, 20 hidden units, networks (compared to for instance Le Cun [LCBD⁺89]) are quite good already. This implies that the data sets are not that difficult to separate in the high dimensional feature space.

The question remains whether the nonlinearity curves reflect the generalisation behaviour of the networks. Let us consider again figure 6.6 and especially the curves for the 100 samples/class case. Both graphs show the trends which can be seen in the test set error curves (right pictures in figure 6.4 and 6.5). For the 15 hidden units network the test set error is slowly decreasing. This is reflected in the corresponding nonlinearity curve (figure 6.6 right). Shortly after starting the training it increases to a maximum at around epoch 50. Af-

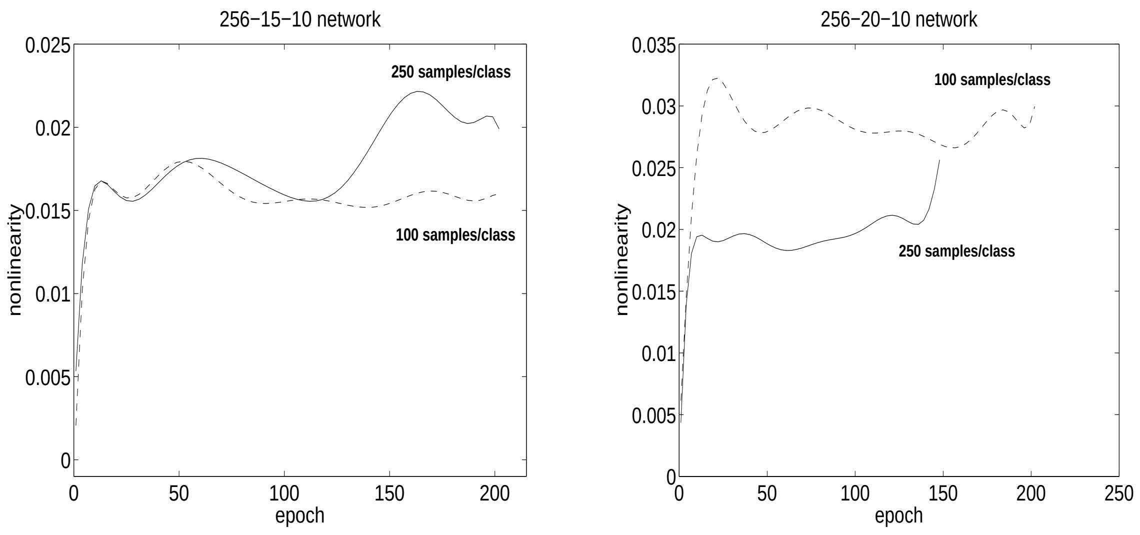


Figure 6.6: *The figures show the nonlinearity development for a network consisting of 15 hidden units (on the left) and 20 hidden units (on the right).*

ter that, it decreases and remains more or less stable, just as the test set error does. The same behaviour is shown in the right graph of figure 6.6. As long as the test set error increases, the nonlinearity also rises. When the error starts to decrease the $\mathcal{N}$ also gets smaller, which is in agreement with the theory presented in chapter 3.

Another interesting graph is depicted in figure 6.7 (left). In this case only the network trained with 25 samples/class shows a nonlinearity. For 100 and 250, no nonlinearity is observed. This indicates that the network is only able to implement an almost linear decision function. A closer look at the nonlinearity curves for 100 and 250 samples/class reveals some small peaks. It is therefore be concluded that the network does not implement a linear function but a slightly nonlinear one which is observed as linear.

6.3.3 Confidence value estimation and rejection

Since no experiments to investigate the symmetry breaking have been done, the next step is to perform a confidence value analysis and use these confidence values as rejection criterion (see chapter 5). Two estimators have been used here: the k-NN and the network output estimator. The logistic estimator was also used, but during analysis it turned out that the estimator suffered from accuracy problems. The underlying routine *loglc* (from the PRTools toolbox) contains a matrix inversion. This caused problems during the estimation of the confidences. Therefore, the results from this estimation are not reliable enough and not included in this section. Another problem is the computation of the confidence values using

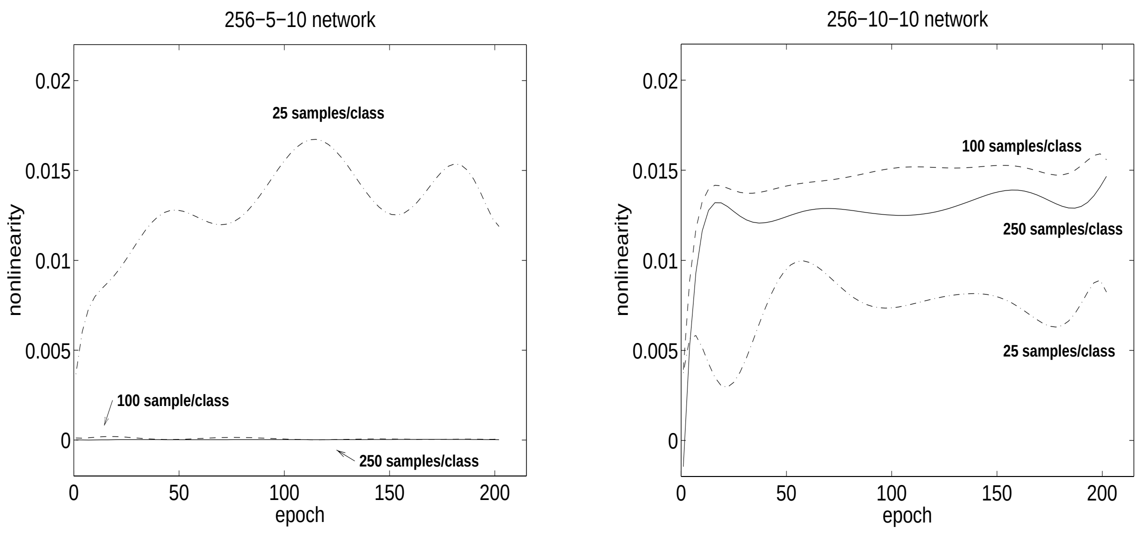


Figure 6.7: *The figures show the nonlinearity development for a network consisting of 5 hidden units (on the left) and 10 hidden units (on the right).*

the *k*-NN estimator. Although the procedure is quite simple, it is computationally expensive. To compute all the confidence values at every time-stamp (which was set to each 25 epochs) can easily take up to 500 hours of computation time on a fast SUN workstation.

Confidence values

Six networks were investigated using the confidence value estimators, three networks with 5 hidden units and three with 20 hidden units. These two cases are of interest since 5 hidden units is obviously too small and 20 hidden units seems to be a satisfactory value. Tables 6.2 and 6.3 summarise the results for two estimators. Note that the measured performance is calculated using the test set of 1,000 samples / class.

When observing both tables it shows that the network output estimator is a poor estimate of the confidence. This is counterintuitive since the performance of the network is quite reasonable, at least in the 20 hidden unit case. From the network performance, one would expect that q_{netout} should perform quite well. The problem, however, is that the dimensionality of the output space is too large. Due to the fact that the networks have 10 output units (one for each class), the output distributions tend to equalise. Although the class label is assigned to the network output having the highest value, the differences between the individual outputs are small. Consequently, none of the outputs will be almost 1 and the rest almost 0. Given the equation for J (equation 5.4 in chapter 5) the squared difference between q and $c(\vec{x}, \omega)$ will be relatively large. Consequently, J will be large. If classifications had been done using two output units, i.e. one unit for the correct class and one unit for not being the correct

Table 6.2: *This table shows J (see equation 5.4) for two estimators of the confidence values, the network output and the k-NN estimator. The results were obtained using the test set. The network under investigation has 5 hidden units.*

	Training set size		
estimator	25	100	250
k-NN			
input	0.1103	0.0664	0.0581
hidden	0.2246	0.2016	0.2372
output	0.2246	0.1306	0.1369
netout	0.3369	0.2967	0.3943
performance	62.53%	53.67%	72.28%

Table 6.3: *This table shows J (see equation 5.4) for two estimators of the confidence values, the network output and the k-NN estimator. The results were obtained using the test set. The network under investigation has 20 hidden units.*

	Training set size		
estimator	25	100	250
k-NN			
input	0.1585	0.0997	0.0775
hidden	0.1256	0.0813	0.0584
output	0.1351	0.0809	0.0656
netout	0.4397	0.4418	0.4494
performance	85.07%	89.00%	91.52%

class, the network outputs would have been more useful.

The *k*-NN estimator works as expected. The *k* was chosen to 16 for 25 samples / class ($\sqrt{25 * 10}$), 32 for 100 samples / class ($\sqrt{100 * 10}$) and 50 for 250 samples / class ($\sqrt{250 * 10}$). The *k*-NN estimator does not suffer from the problem sketched above since it uses the "two-class" approach. Only the probability of belonging to the class is calculated, i.e. $P(S(\vec{x}) = \omega|\vec{x})$. The probability of belonging to another class is the complement, i.e. $1 - P(S(\vec{x}) = \omega|\vec{x})$.

Networks consisting of 5 hidden units are too small to reliably classify the samples. J is relatively high, indicating that many samples are assigned a confidence value of approximately 0.5. The larger the network, and the larger its capacity, the smaller J becomes. This behaviour is expected since larger networks are able to capture more of the data set than smaller networks. Also a shift from the best estimator from the input space to the output space can be noticed. For 5 hidden units the best estimator in the 100 samples / class case is in the input, for 20 hidden units the output unit space is better. The next section will present the details of the consequences for the rejection curves.

Rejection

The last experiments involved the use of confidence values for the rejection. As explained in chapter 5 the confidence values can be used to improve the classifier's performance. Based on the results in tables 6.2 and 6.3 one can select one of the estimators to assist in rejection. The experiments presented here involved all the *k*-NN estimators in different layers of the network and the network outputs. In figures 6.8, 6.9 and 6.10 the rejection curves for the network with 5 hidden units are depicted.

As expected from table 6.2 the *k*-NN estimator gives the best rejection performance if its confidence values are used. From the right graphs in figures 6.8, 6.9 and 6.10 it can be concluded that the netout estimator always performs worse than the best *k*-NN estimator, although it shows in general the same kind of behaviour as the *k*-NN estimator. The most significant performance is gained when rejecting about 25% of the samples (the results have not been detailed out to the level of individual classes). The left graphs (figure 6.8, 6.9 and 6.10) show that the estimators for the hidden and output layer usually stay together and the input estimator is better. How does this behaviour change when using a larger number of hidden units? Figures 6.11, 6.12 and 6.13 show the results for networks with 20 hidden units.

Remarkable when observing the graphs is that the network output estimator is not significantly worse than the best *k*-NN estimator. Sometimes the netout estimator is even better. This indicates that the values of J in table 6.3 are not as bad as expected. We have already explained why the Js for the netout estimator are high. For multi-class cases, J is probably

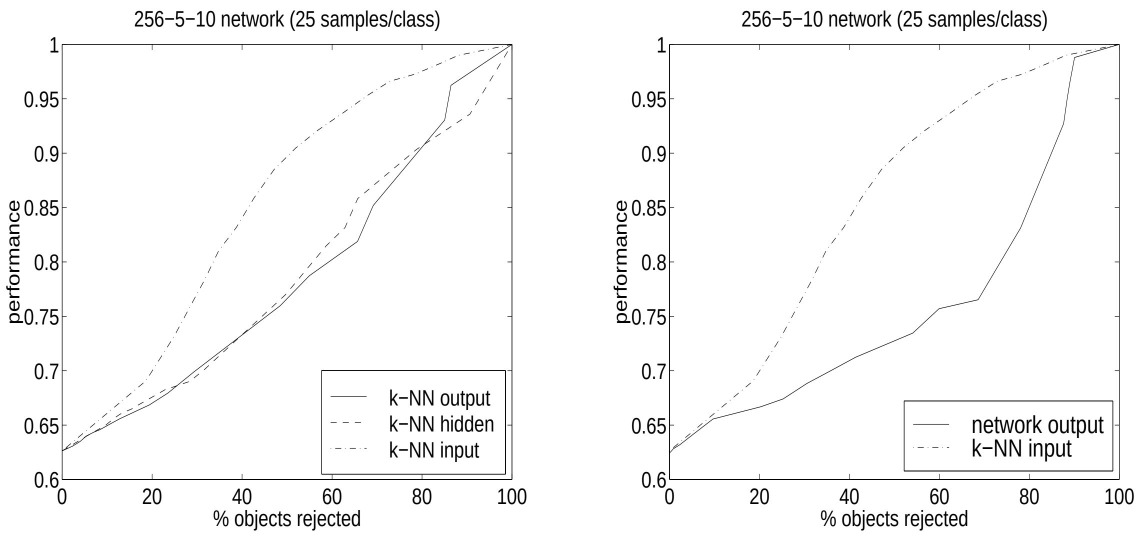


Figure 6.8: *The graph on the left shows the rejection curve when using the confidence values of the k-NN estimator. The graph on the right shows the network output estimator and the best k-NN estimator from table 6.2. This network with 5 hidden units was trained with 25 samples/class.*

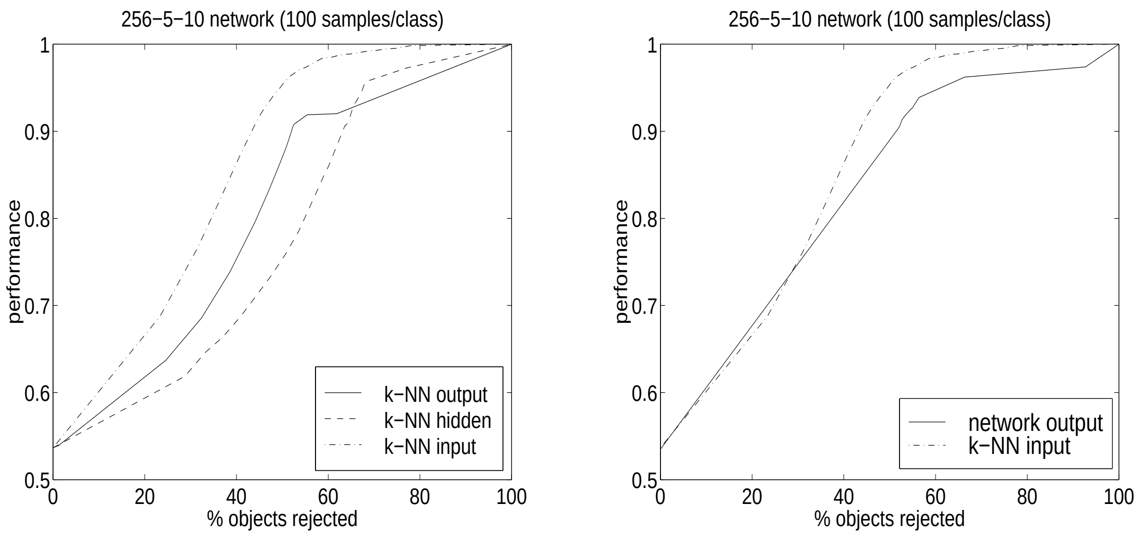
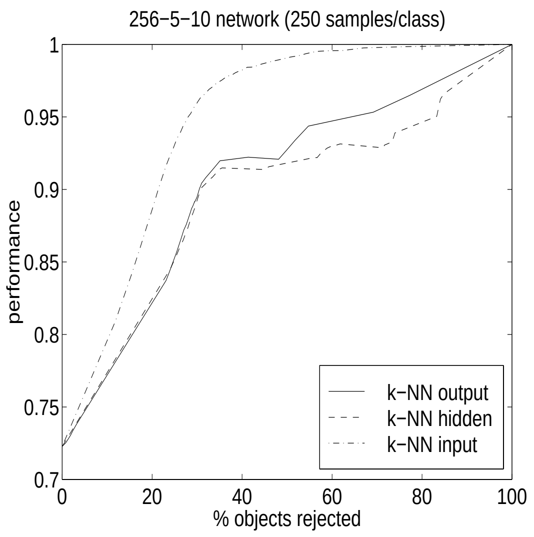


Figure 6.9: *The graph on the left shows the rejection curve when using the confidence values of the k-NN estimator. The graph on the right shows the network output estimator and the best k-NN estimator from table 6.2. This network with 5 hidden units was trained with 100 samples/class.*



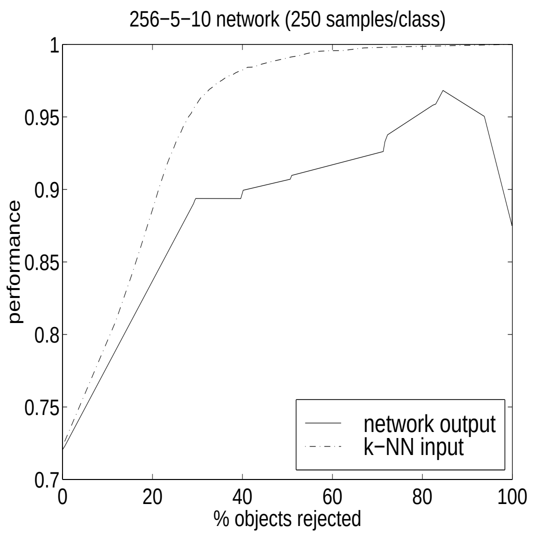
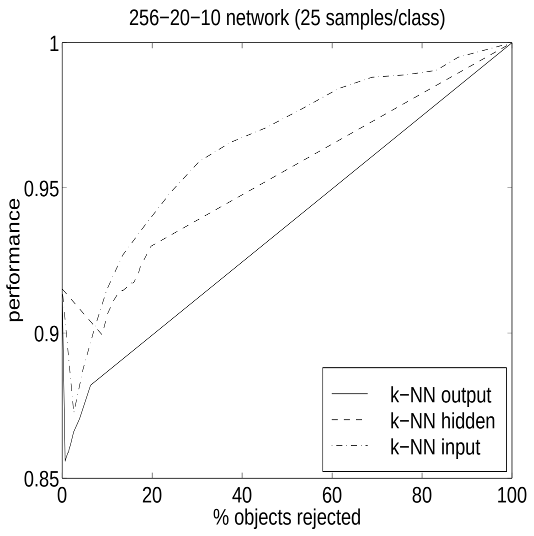


Figure 6.10: *The graph on the left shows the rejection curve when using the confidence values of the k-NN estimator. The graph on the right shows the network output estimator and the best k-NN estimator from table 6.2. This with 5 hidden units network was trained with 250 samples/class.*



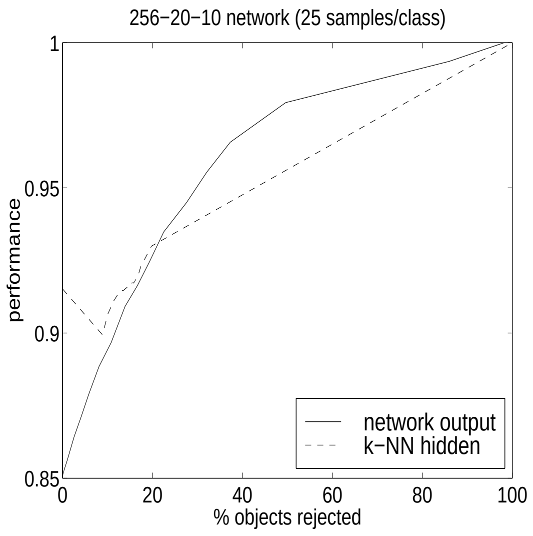


Figure 6.11: *The graph on the left shows the rejection curve when using the confidences of the k-NN estimator. The graph on the right shows the network output estimator and the best k-NN estimator from table 6.3. This network was trained with 25 samples/class.*

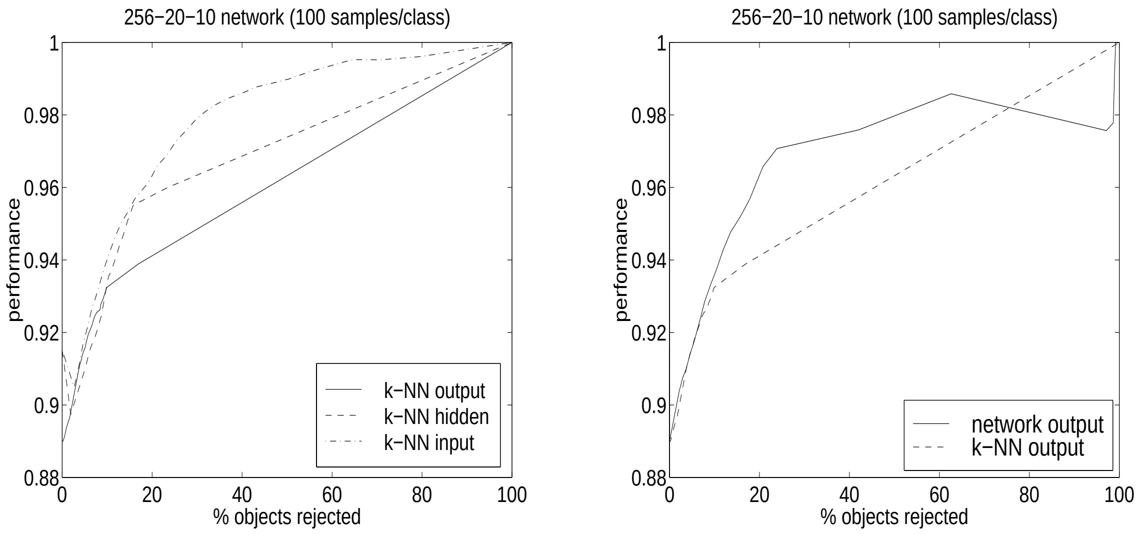


Figure 6.12: *The graph on the left shows the rejection curve when using the confidences of the k-NN estimator. The graph on the right shows the network output estimator and the best k-NN estimator from table 6.3. This network was trained with 100 samples/class.*

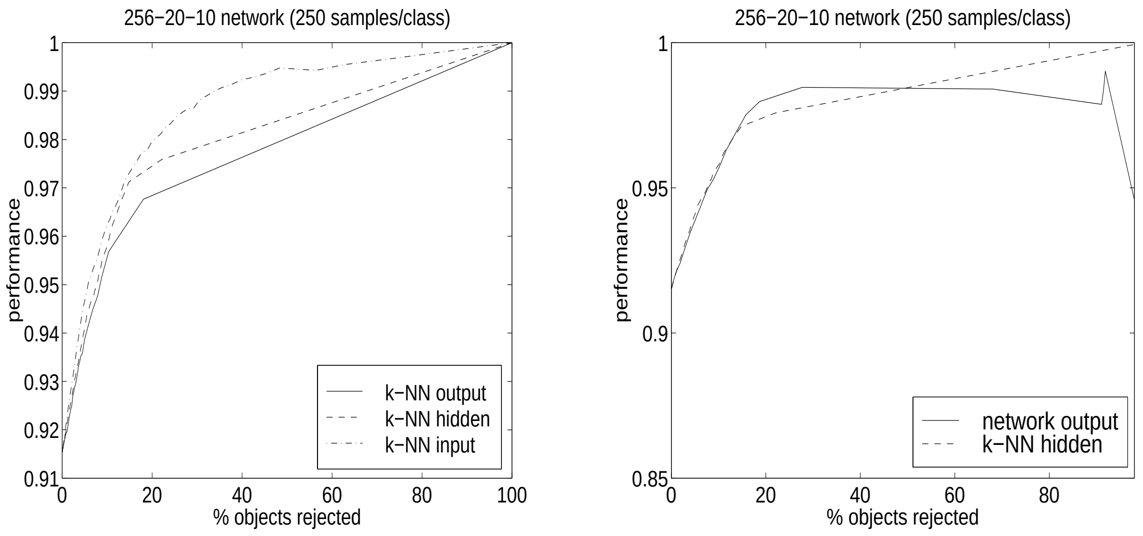


Figure 6.13: *The graph on the left shows the rejection curve when using the confidences of the k-NN estimator. The graph on the right shows the network output estimator and the best k-NN estimator from table 6.3. This network was trained with 250 samples/class.*

on the average much higher and even close to 0.5. This phenomenon was also observed by Tholen [Tho95]. Remarkable is also the fact that the best *k*-NN estimator does not have the steepest curve when starting to reject. It is even the case that the best estimator derived from table 6.3 is usually worse than others with respect to the "slope" property. A reason for this may be the fact that the curves are computed on the networks having the lowest test set error. It may be beneficial to accept a slightly higher test set error in order to have a better rejection behaviour. Since the costs of re-examining a false rejection may be lower than accepting one.

6.4 Discussion

In this chapter it was shown how the methods introduced in chapters 3 and 5 behave in more practical problems. The methods studied in chapter 4 were not used since this caused computational problems. This immediately shows the shortcoming of the visualisation method. Due to the huge feature space, combined with a large number of hidden units, an acceptable result within a reasonable amount of time was not possible. Some unreported experiments done to classify chromosome banding profiles show that it takes at least two months to come to a sensible result.

In general it can be concluded that a fair performance for digit recognition can be achieved using a relatively simple network. Although the performance is less than the best possible solution it does not take too much training time. More complex networks, i.e. a larger number of hidden units, will only lead to longer training times. An alternative is to use rejection or other network architectures [LCBD$^+$89, LCS91, dR96]. Although with rejection a human operator is required or possibly a multi-stage classifier. If more hidden units are used, the better the samples are classified, although 20 hidden units seems to be an upperbound.

Given the dimensionality of the data set it was expected that the network would show a large and quick rise of the nonlinearity. From figures 6.6 and 6.7 it can be observed that the nonlinearity resembles the values which were presented for the two dimensional data sets. Apparently the intrinsic dimensionality of the data set is such that it can be separated using a almost linear classifier. The behaviour of the nonlinearity is in accordance with the theory and examples presented in chapter 3. Although no direct statement in that chapter was made about the behaviour of the nonlinearity measure in high dimensions it seems to operate quite well. This due to the fact that the definition of the nonlinearity measure (definition 4) depends on the classification made by the classifier. The classifier, however, may be sensitive to the dimensionality which then will be reflected in the nonlinearity behaviour.

The confidence value estimators caused more problems. The network output estimator suffers from the fact that the dimensionality of the output space is too high. This “curse of dimensionality” was already observed as a problem in the experiments done in chapter 5. Since the output values of the individual units are used for probability estimation, there is no clear preference for a particular unit. This lead to the conclusion that the network output confidence values were unsuited for rejection. However, the rejection curves in figures 6.8 till 6.13 show that this is not the case in our experiments. The logistic estimator is unreliable since it uses a linear estimation procedure which requires a matrix inversion. This matrix inversion fails due to numerical problems for higher dimensions. The Parzen estimator has not been used since earlier experiments showed that it fails in high dimensional feature spaces. As already observed in the experiments of chapter 5 either the k-NN estimator or network output estimator are the best estimators for the confidence and thereby a basis for a rejection rule.

Although we have achieved relatively good performances for the classification of handwritten digits, one should realise we have used a very clean data set. No noise or other disturbances were present.

Chapter 7

Conclusions

The general theme of this thesis was the analysis and study of the generalisation during the training of feed forward backpropagation trained networks. We started with theoretical frameworks for generalisation (chapter 2) as they are found in the current literature. These frameworks were used as a starting point to introduce several new tools in the analysis of the training behaviour. Each of these tools can be used in the training cycle of a network, such that it enables the study and analysis of the generalisation behaviour. In order of introduction, the following aspects were studied:

Chapter 3: Nonlinearity. The nonlinearity measure can be used to quantify the shape of the decision function implemented by a classifier in relation to the data is has to classify. During training the decision function implemented by a classifier adapts to the training data. Starting from a linear solution the decision function becomes more and more nonlinear. Overtrained classifiers have adapted too much to the noise in the data set. Therefore, it does not generalise anymore. The chapter showed that the nonlinearity measure can be used to detect such non-generalising situations. Non-generalising classifiers will in general have a nonlinearity which is too high compared to the optimal classifier. Furthermore, the nonlinearity measure allows a comparison between classical and neural network classifiers.

Chapter 4: Symmetry breaking. An important aspect of using neural network classifiers is the choice of their architecture. By selecting the appropriate number of hidden units (other aspects of the network architecture selection were not addressed in this thesis) a network is able to learn faster and/or better. using more hidden units than theoretically necessary introduces redundancy in the network. This redundancy helps during training, networks will in general converge much faster. During training the hidden units of a network tend to follow some trajectory in a weights space. At certain time steps, these trajectories start to differ the network breaks its symmetry. This symmetry breaking is a known phenomenon in the literature, however, it was not yet well-visualised. The well-known self-organising map was used to map the weights of the hidden units on a 2-dimensional plane such that their behaviour during training can be displayed. It was concluded that the symmetry breaking indeed occurs and helps the network during training (redundancy helps). Too much symmetry breaking (i.e. no redundancy left) in a network might indicate that it is overtrained.

Chapter 5: Confidence values. A last aspect which was studied, is the reliability of trained neural network classifiers. After training a classifier one is interested in how reliable are its classifications. By the introduction of confidence values the a posteriori probabilities of the sample under investigation can be estimated. Several methods for the estimation of the confidences were introduced and analysed: *network output*, *k-NN* and *logistic*. Furthermore, these methods can be applied in different parts of the network: *input*, *hidden* and *output* layer. Also a method was introduced to generate an artificial "test set" by using *k*-NN data generation. It was concluded that confidence values can be used to indicate whether a neural network is adequately trained or not. Furthermore, the confidences can be used as a rejection method to improve the quality of a classifier, or to build new multi-stage classifiers. The *k*-NN generation method was also shown to be a valid method to generate an artificial test set.

Finally in chapter 6 the validity of the introduced methods was assessed for a practical problem. Since each of the chapters has its own discussion and conclusions the remainder of this chapter is devoted to more general topics in neural network and pattern recognition research.

One of the main aims in pattern recognition and neural network research is to develop classifiers which can be used in practice. Within this view the currently available neural classifiers constitute a problem, in the sense that they tend to have too many parameters to adjust. The operator who is using the classifier is not interested in which parameters to adjust. He/she just wants to have a classifier which optimally performs the task in terms of misclassifications. The automatic choice of the best set of parameters for a classifier has not been addressed in this thesis since it is beyond its scope, although it is implicitly present in the research material presented in the various chapters. It is, however, an important issue and therefore some remarks about it are appropriate.

Automatic selection of the best set of parameters, e.g. the optimal learning rate or the best number of hidden units, requires a thorough analysis of the data set which has to be classified. On the other hand a thorough understanding of the classifier is also needed. This problem of data set and classifier complexity was partially addressed in [HDK97]. Ideally, a data set and a classifier should be characterised by a simple relation or some magic number.

To find such magic numbers is a very difficult problem. For classifiers several options are available. In the case of a neural network one could use the VC-dimension, or a tighter bound such as the effective capacity. These dimensions, however, are hard to estimate. A much easier option is to relate the characteristics to the number of hidden units. For instance a network consisting of 2 hidden units is, at most, capable of implementing a quadratic decision function (see chapter 1 and 4). However, if a network is capable of implementing a function of a certain degree, it is not guaranteed that it will find it. The learning algorithm restricts the possible set of implemented functions, therefore it should also be taken into account.

The same kind of situation holds for a data set. Given a certain data set, one would like to be able to state what kind of classifier is needed. Consequently, some kind of separability analysis has to be done. That is, investigate what the complexity of the data set is. If a data set can be separated by a linear function, it can also be separated by a function with a higher degree. In any case the classifier with the least complexity and the smallest error should be used. Also the number of samples in the data set should be taken into account. The more samples available, the better an estimation of the densities in the data set can be done.

The problem, however, is that a classifier and a data set cannot be viewed as separate items. In order to select an appropriate classifier to solve a classification problem, one has to perform both data analysis and classifier analysis. Within the context of this thesis the following steps might be taken [HDK97]:

1. Perform data analysis to gain insight into the structure and complexity of the data under investigation. See Hoekstra et al. [HDK97].

2. Select a neural network architecture, or a class of architectures to train using the data. For instance by using the visualisation method explained in chapter 4.

3. Find the best classifier(s) by monitoring its nonlinearity during training and perform a stability analysis as in chapter 3 and [HDK97].

4. After training perform a reliability analysis using the confidence values, as explained in chapter 5.

These steps do not guarantee that the best classifier will be found. They are, however, a structured mechanism to find an appropriate classifier. Moreover, the steps might be executed with no intervening operator. This allows an automatic selection of an appropriate classifier.

A last issue which we want to address in these conclusions is the problem of neural network implementation. The experiments presented in this thesis were performed in Matlab and ANN/SPRLIB and performed on SUN sparcstations. Although these workstation are fast, experiments usually take a few weeks sometimes even months. This is a severe problem of the application of neural networks and their tools. No user will accept such long training times when using neural networks. Trained neural networks are usually no problem, since in principle feed forward networks are easily implemented by a series of matrix multiplications. The procedure, however, does not allow any re-training. The most important problems when implementing and training neural networks are:

- How much flexibility should one offer in a software environment? Too much flexibility will lead to superfluous code but allows a wide variety of architectures. Less flexible code allows a faster and more efficient implementation.
- Software engineering is a process in which bugs are still unavoidably introduced into the software. In practice it is sometimes hard to find out whether an unexpected result is due to a software bug or just a false hypothesis.
- Given neural network software, the network training takes a fairly large amount of time, sometimes months even when a fast computer is available. In general, code written in C (e.g. ANN / SPRLIB) will execute much faster than code written in Matlab.
- The user interface of neural network software is, in general rather, crude. This hampers the use of software in practical situations in which one is just interested in applying a neural network method in an efficient manner.

Naturally more problems arise when dealing with neural networks, but this list shows that using neural networks is not a trivial task. This does not imply that neural networks should not be used. They are very powerful machines when used in the correct environment. From the research presented in this thesis we have gained more insight into the working of a neural network. This enables us to apply neural networks with more knowledge and allows a better control.

Bibliography

[AB66] A. G. Arkadjew and E. M. Brawerman. *Zeichenerkennung und maschinelles lernen*. R.Oldenbourg:Munich and Vienna, 1966.

[And82] J. A. Anderson. *Logistic discrimination*, volume 2 of *Handbook of Statistics*, pages 169–191. North-Holland, 1982.

[Ann94] A. J. Annema. *Analysis, Modeling and Implementation of Analog Integrated Neural Networks*. PhD thesis, Twente University, 1994.

[Bac95] E. Backer. *Computer Assisted Reasoning in Cluster Analysis*. Prentice-Hall, Englewood Cliffs, 1995.

[Bau88] E.B. Baum. On the capabilities of multi-layer perceptrons. *Journal of complexity*, pages 193–215, 1988.

[BH89] E. B. Baum and D. Haussler. What size net gives valid generalization ? *Neural computation 1*, pages 151–160, 1989.

[Bis95] C. M. Bishop. *Neural networks for pattern recognition*. Clarendon Press, Oxford, 1995.

[BJ90] R. Beale and T. Jackson. *Neural computing, an introduction*. Adam Hilger, Bristol, England, 1990.

[BLV90] A. Barnas and A. La Vigna. Convergence of Kohonen's learning vector quantization. In *IJCNN, International Joint Conference on Neural Networks*, pages 21–26. IEEE, 17-21 June 1990. vol 3.

[CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 3(20):273–297, 1995.

[Das91] B.V. Dasarathy, editor. *Nearest neighbor (NN) Norms: NN pattern classification techniques*. IEEE computer society press tutorial, 1991.

[DB93] H. Demuth and M. Beale. *Neural Network toolbox for use with Matlab*. The Mathworks Inc., 1993.

[DdR97] R.P.W. Duin and D. de Ridder. Neural network experiences between perceptrons and support vectors. In *Proceedings of the 8th British Machine Vision Conference*, pages 590–599, 1997.

[DdRT97] R.P.W. Duin, D. de Ridder, and D.M.J. Tax. Featureless classification. In *Proceedings of the 1st IAPR TC1 workshop on statistical techniques in pattern recognition*, pages 37–42, 1997.

[Dev88] L. Devroye. Automatic pattern recognition: A study of the probability of error. *IEEE Transactions on pattern analysis and machine intelligence*, 10(4):530–543, July 1988.

[DH73] R. O. Duda and P. E. Hart. *Pattern classification and scene analysis*. John Wiley and sons, 1973.

[DLC91] J. S. Denker and Y. Le Cun. Transforming neural-net output levels to probability distribution. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Advances in Neural information Processing Systems 3*, pages 853–859. Morgan Kaufmann, 1991.

[dR96] D. de Ridder. Shared weights neural networks in image analysis. Master's thesis, Pattern Recognition Group, Delft University of Technology, March 1996.

[Dui78] R. P. W. Duin. *On the accuracy of statistical pattern recognizers*. PhD thesis, Delft University of Technology, 1978.

[Dui93a] R. P. W. Duin. Nearest neighbor interpolation for error estimation and classifier optimization. In *Proceedings of the 8th SCIA Tromsø*, 1993.

[Dui93b] R. P. W. Duin. Superlearning capabilities of neural networks ? In *Proceedings of the 8th SCIA*, pages 547–554. Norwegian Society for Image Processing and Pattern Recognition, May 25-28 1993.

[Dui95a] R. P. W. Duin. *PRTools a Matlab toolbox for pattern recognition*. Pattern Recognition group, Delft University of Technology, October 1995.

[Dui95b] R. P. W. Duin. Small sample size generalization. In G. Borgerfors, editor, *Proceedings of the 9th SCIA*, pages 957–964, June 6-9 1995.

[Fuk90] K. Fukanaga. *Introduction to statistical pattern recognition*. Academic Press, 1990.

[Fun89] K-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989.

[Has94] S. Hashem. Optimal linear combinations of neural networks. *Neural Networks*, October 1994. submitted, current status unknown.

[HD93] A. Hoekstra and M. F. J. Drossaers. An extended Kohonen feature map for sentence recognition. In *Proceedings of the ICANN '93*, pages 404–407, 1993.

[HD95] A. Hoekstra and R. P. W. Duin. Exploring the capacity of simple neural networks. In J. van Katwijk, J.J. Gerbrands, M.R. van Steen, and J.F.M. Tonino, editors, *Proceedings of the first annual conference of the ASCI*, pages 56–62, May 16-18 1995.

[HD96] A. Hoekstra and R. P. W. Duin. On the nonlinearity of pattern classifiers. In *Proceedings of the 13th ICPR, Vienna*, pages 271–275. IEEE Computer Society Press, Los Alamitos, 1996. Vol. 3, track D: Parallel and Connectionist Systems.

[HD97] A. Hoekstra and R. P. W. Duin. Investigating redundancy in feed forward neural classifiers. *Pattern Recognition Letters*, 18, 1997. special issue: Pattern Recognition in Practice V.

[HDK97] A. Hoekstra, R. P. W. Duin, and M. A. Kraaijveld. *Neural Network Systems Techniques and Applications*, chapter Neural Networks applied to data analysis. Academic Press, 1997. In press.

[HHS94] T.K. Ho, J.J. Hull, and S.N. Srihari. Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(1):66–75, January 1994.

[HIVK93] J. B. Hampshire II and B. V. K. Vijaya Kumar. Differential theory of learning for efficient neural network pattern recognition. In *Proceedings of the 1993 SPIE International Symposium on Optical Enigneering and Photonics in Aerospace and Remote Sensing*, april 1993.

[HKdRS96] A. Hoekstra, M. A. Kraaijveld, D. de Ridder, and W. F. Schmidt. *The Complete SPRLIB & ANNLIB*. Pattern Recognition Group, Faculty of Applied Physics, Delft University of Technology, 1996.

[HKP91] J. A. Hertz, A. S. Krogh, and R. G. Palmer. *Introduction to the theory of neural computation*. Sante Fe institute studies in the sciences of complexity. Addison-Wesley, Reading:Mass, 1991.

[HM94] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989 – 993, 1994.

[HNdR96] A. Hoekstra, H Netten, and D. de Ridder. A neural network applied to spot counting. In *Proceedings ASCI'96, 2nd annual conference of the advanced school for computing and imaging*, pages 224–229, June 5-7 1996.

[Hol96] A. Hole. Vapnik-Chervonenkis generalization bounds for real valued neural networks. *Neural Computation*, 8(6):1277–1299, 1996.

[Hop82] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume USA 79, pages 2554–2558, 1982.

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2:359–366, 1989.

[HTD96] A. Hoekstra, S. A. Tholen, and R. P. W. Duin. Estimating the reliability of neural network classifications. In *Proceedings of the ICANN 96, Bochum Germany*, pages 53–58, 1996.

[Inc96] The Mathworks Inc. *Matlab: The language of technical computing, version 5*, 1996.

[JC82] A. K. Jain and B. Chandrasekran. *Handbook of statistics*, volume 9, pages 835–855. North Holland, 1982.

[JW78] A. K. Jain and W. G. Waller. On the optimal number of features in the classification of multivariate gaussian data. *Pattern recognition*, 10:365–374, 1978.

[KC71] L. N. Kanal and B. Chandrasekaran. On dimensionality and sample size in statistical pattern recognition. *Pattern recognition*, 3:225–234, 1971.

[KD94] M. A. Kraaijveld and R. P. W. Duin. The effective capacity of multilayer feedforward network classifiers. In *Proceedings of the 12^{th} ICPR*, pages B–99 – B–103, 1994.

[KHD96] M. Kittler, M. Hatef, and R. P. W. Duin. Combining classifiers. In *Proceedings of the 13th ICPR, Vienna*, pages 897–901. IEEE Computer Society Press, Los Alamitos, 1996. vol. 2, track B: Pattern Recognition and Signal Analysis.

[KHKL95] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. *SOMPAK The Self-Organising Map program package*. Laboratory of Computer and Information Science, Helsinki University of Technology, Rakentajanaukio 2 C, SF-02150 Espoo, FINLAND, version 3.1 edition, April 7 1995.

[KK82] L. N. Kanal and P. R. Krishnaiah. *Handbook of statistics 2: Classification pattern recognition and reduction of dimensionality*. North-Holland, 1982.

[KMJ95] M. A. Kraaijveld, J. Mao, and A. K. Jain. A nonlinear projection method based on Kohonen's topology preserving maps. *IEEE transactions on neural networks*, 6(6):548–559, 1995.

[Koh89] T. Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Heidelberg, 3rd edition, 1989.

[Koh90] T. Kohonen. The self-organizing map. In *Proceedings IEEE*, pages 1464–1480, September 1990. vol 78, no 9.

[Kra93] M. A. Kraaijveld. *Small sample behavior of multi-layer feedforward network classifiers: theoretical and practical aspects*. PhD thesis, Delft University of Technology, 1993.

[LCBD$^+$89] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1:541 – 551, 1989.

[LCS91] Y. Le Cun and S. A. Solla. Constrained neural networks for pattern recognition. In *Neural networks: Concepts, Applications and Implementations*. Prentice Hall, 1991.

[Lev44] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly journal of applied mathematics*, II(2):164–168, 1944.

[Moo92] J. E. Moody. The effective number of parameters: An analysis of generalization and regularization in nonlinear systems. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 847 – 854. Morgan Kaufmann Publishers, San Mateo: CA, 1992.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of mathematical biophysics*, 5:115–133, 1943.

[MP69] M. L. Minsky and S. A. Papert. *Perceptrons*. Cambridge: MIT Press, 1969.

[MST94] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood series in artificial intelligence. Ellis Horwood, 1994.

[Rau93] S. Raudys. On shape of pattern error function, initializations and intrinsic dimensionality in ann classifier design. *INFORMATICA*, 3-4:360–363, 1993.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In Rumelhart, D.E. and McClelland, J.L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume I. MIT Press, Cambridge, MA, 1986.

[Rip96] B. Ripley. *Recognition and Neural Networks.* Cambridge university press, 1996.

[RSM91] H. Ritter, K. Schulten, and T. Martinetz. *Neural computation and self-organizing maps: an introduction.* Addison-Wesley, 1991.

[Sch93] W. F. Schmidt. *Neural pattern classifying systems.* PhD thesis, Delft University of Technology, 1993.

[Sch97] B. Schölkopf. *Support vector learning.* PhD thesis, Technical University Berlin, 1997.

[Sim90] P. K. Simpson. *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations.* Pergamon Press: New York, 1990.

[SR86] T.J. Sejnowski and C. R. Rosenberg. NETtalk: a parallel network that learns to read aloud. Technical Report JHU/EECS-86/01, The Johns Hopkins University Electrical Engineering and Computer Science, 1986.

[SRD98] M. Skurichina, S. Raudys, and R. P. W. Duin. K-nearest neighbours directed noise injection in multilayer perceptron training. *IEEE Transactions on Neural Networks*, 1998. submitted.

[TDB97] D.M.J. Tax, R.P.W. Duin, and van M. Breukelen. Comparison between product and mean classifier combination rules. In *Proceedings of the 1st IAPR TC1 workshop on statistical techniques in pattern recognition*, pages 165–170, 1997.

[TdRD97] D.M.J. Tax, D. de Ridder, and R. P. W. Duin. Support vector classifiers: a first look. In H.E. Bal, H. Corporaal, P.P. Jonker, and J.F.M. Tonino, editors, *Proceedings of the third annual conference of the Advanced School for Computing and Imaging*, pages 253–258, June 1997.

[Tho95] S. A. Tholen. Confidence values for neural network classifications. Master's thesis, Delft University of Technology, April 1995.

[Vap82] V. Vapnik. *Estimation of dependencies based on empirical data.* Heidelberg: Springer-Verlag, 1982.

[Vap95] V. Vapnik. *The nature of Statistical Learning Theory.* Heidelberg: Springer Verlag, 1995.

[vBDT97] M. van Breukelen, R.P.W. Duin, and D.M.J. Tax. Combining classifiers for the recognition of handwritten digits. In *Proceedings of the 1st IAPR TC1 workshop on statistical techniques in pattern recognition*, pages 13–18, 1997.

[VKA94] V. Roychowdhury, K-Y Siu, and A. Orlitsky, editors. *Theoretical Advances in Neural Computation and Learning*. Kluwer Academic Publishers, Dordrecht, 1994.

[Vog94] A. C. Vogtländer. The learning behaviour of multilayer networks used as classifiers. Master's thesis, Faculty of Applied Physics, Delft University of Technology, 1994.

[vOY78] P.J. van Otterloo and I.T. Young. A distribution-free geometric upper bound for the probability of error of a minimum distance classifier. *Pattern Recognition*, 10:281–286, 1978.

[Wer74] P. Werbos. *Beyond regression: new tools for prediction and analysis in the behavioural sciences*. PhD thesis, Harvard University, 1974.

[WG90] C. L. Wilson and M. D. Garris. Handprinted character database 2, april 1990. National Institute of Standards and Technology; Advanced Systems division.

[WG92] C. L. Wilkinson and M. D. Garris. Handprinted character database 3, february 1992. National Institute of Standards and Technology; Advanced Systems division.

[WH60] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention record*, pages 96–104. New York: IRE, 1960. part 4.

[Wol93a] D. H. Wolpert. On bias plus variance. Technical Report SFI TR 95-007, The Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, NM, 87501, 1993.

[Wol93b] D. H. Wolpert. On overfitting avoidance as bias. Technical Report SFI TR 92-03-5001, The Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, NM, 87501, 1993.

[Wol94] Wolpert, D.H., editor. *The mathematics of generalization*. Proceedings of the SFI/CNLS workshop on formal approaches to supervised learning, November 1994.

[ZR95] H. Zhu and R. Rohwer. Bayesian invariant measurements of generalisation for continous distributions. Technical Report NCRG/4352, Neural computing research group, Dept. of computer science and applied mathematics, Aston university, Birmingham B4 7ET, UK, august 1995.

Samenvatting

Generalisation in Feed Forward Neural Classifiers

De laatste jaren is het gebruik van neurale netwerken in de patroonherkenning gemeengoed geworden. Neurale netwerken zijn krachtige classificatoren in vergelijking tot de klassieke algoritmes als naaste nabuur methodes. De netwerk algoritmen die gebruikt worden zijn in staat om aan de hand van een eindige, en qua omvang een relatief kleine, set van voorbeelden toch een goede classificator te vinden. Met name dat vermogen, ook wel generalisatie genoemd, is vanuit patroonherkenningsoogpunt interessant omdat een grote set van parameters geschat wordt gebaseerd op vanuit de theorie gezien te weinig data. In dit proefschrift wordt het generalisatiegedrag van neurale netwerken bestudeerd. Met name vragen hoe dit gedetecteerd kan worden en welke invloeden daarbij een rol spelen worden beantwoord.

Om deze vragen te kunnen beantwoorden is er allereerst een goed begrip van het begrip "generalisatie" nodig, zodat de resultaten van de diverse gebruikte technieken vergeleken kunnen worden. Daartoe wordt een operationele definitie van *generalisatie* geïntroduceerd, namelijk het verwachte aantal fouten wat een getrainde classificator maakt op een set van testvoorbeelden. De set van classifiactoren waartoe het proefschrift zich beperkt, is de klasse van "feed forward backpropagation" getrainde neurale netwerken en de klassieke patroonherkenners als de k-naaste nabuur regel, en lineaire en kwadratische classificatoren.

Als eerste moet meer inzicht in het gedrag van neurale netwerken worden verkregen. Daarna kan er een maat toegepast worden die voor beide geldig is, om zo klassieke en neurale classificatoren met elkaar te vergelijken. Door gebruik te maken van de niet-lineariteit van een classificator, kan inzicht worden verkregen in het generalisatiegedrag van neurale classificatoren. De niet-lineariteitsmaat zegt iets over de mate waarin een classificator zich aan de dataset aangepast heeft. Hoe beter de classificator zich aanpast aan de data, des te kleiner de generalisatiefout. Echter, een te grote aanpassing aan de data, oftewel een te grote

niet-lineariteit, geeft een slecht generaliserende classificator. Door de niet-lineariteit te volgen tijdens het trainen kan men inzicht in het generalisatiegedrag verkrijgen. De definitie van de niet-lineariteit van een classificator is zodanig dat het ook toepasbaar is op niet-neurale classificatoren. Hierdoor is een vergelijking van klassieke en neurale classificatoren mogelijk. Het blijkt dat neurale netwerken in eerste instantie vanuit een lineaire toestand beginnen en zich langzamerhand niet lineair aanpassen. De aanpassing is zelfs sterker (dus uiteindelijk resulterend in een hoge niet-lineariteit) dan voor een klassieke classificator. Dit zou kunnen betekenen dat neurale classificatoren een grotere effectieve capaciteit hebben dan klassieke.

Een andere factor, die van invloed is op het generalisatiegedrag, is de architectuur van het netwerk. In dit geval wordt onder architectuur het aantal verborgen (*hidden*) neuronen bedoeld. Naarmate het aantal verborgen neuronen groter is dan noodzakelijk, maar niet tè groot, zal een netwerk sneller te leren. Dit is een gevolg van de redundantie die in het netwerk aanwezig is. Deze redundantie helpt. Redundantie zorgt ervoor dat neuronen clustergedrag vertonen. Aan het begin van het leerproces hebben alle neuronen ongeveer dezelfde functie en naarmate het leerproces verloopt zullen deze zich specialiseren. Deze specialisatie, ook bekend als symmetrie breking (*symmetry breaking*), kan worden gevisualiseerd met behulp van een projectie techniek. Deze techniek beeldt de hoog-dimensionale ruimte, waarin de neuronen zich bewegen, af op een 2-dimensionaal vlak. In dit vlak kan men vervolgens de trajecten die afgelegd worden tekenen, om zo inzicht te krijgen in het trainingsgedrag.

De laatste factor die (in dit proefschrift) onderzocht is, is de betrouwbaarheid van neurale netwerk classificaties. Na een trainings-proces is men geïnteresseerd in de mate waarin een netwerk de data goed heeft leren verwerken. Hiertoe kan men aan de uitgangen van het netwerk à posteriori kansen gaan schatten en uiteindelijk deze schattingen gebruiken om betere netwerken te gaan bouwen dan wel voorbeelden te verwerpen (*rejection*). Het schatten van deze kansen is mogelijk door gebruik te maken van betrouwbaarheidsschatters. Deze schatters kunnen op diverse manieren geschat worden aan de hand van drie technieken: de netwerk uitgangen, de naaste nabuur methode en de logistische schatting. Aan de hand van deze schatters wordt nagegaan hoe betrouwbaar het netwerk classificeert. Een probleem is echter dat de schatters een onafhankelijke testset nodig hebben. Deze set kan uiteraard maar één keer gebruikt worden en dient beschikbaar te blijven voor de beoordeling van het trainingsproces.

Om dit probleem te ondervangen is gebruik gemaakt van data generatie. Aan de hand van de kenmerken van de training set wordt een nieuwe set gegenereerd. Deze blijkt als vervanger van een test set te kunnen dienen.

Concluderend kan dan ook gesteld worden, ook na toepassing van de technieken in een relatief eenvoudig classificatieprobleem, dat de introductie van de diverse technieken en onderzoeken een beter begrip verkregen is van generalisatie in neurale netwerken. Met name de niet-lineairiteitsmaat is een vooruitgang omdat hierdoor klassieke en neurale classificatoren vergeleken kunnen worden met betrekking tot hun gedrag op een dataset.

Aarnoud Hoekstra, 1998

Dankwoord

Vier jaar, het lijkt een lange tijd voor je promotieonderzoek. Maar voor je het weet zit je te ploeteren op de tekst en indeling van het proefschrift. Het uiteindelijke resultaat is natuurlijk niet alleen de prestatie van ondergetekende, maar een samenspel van factoren. Een dankwoord stelt mij gelukkig in de gelegenheid om diegenen te bedanken met wie ik de afgelopen vier jaar heb doorgebracht.

De ruimte is beperkt dus ik zal niet iedereen stuk voor stuk op gaan sommen. Allereerst wil ik iedereen die werkzaam is (of geweest) binnen de vakgroep PH bedanken voor zijn of haar steun, al dan niet gemeend, bij mijn onderzoek. Met name de dagelijkse koffiepauzes om half elf waren een bron van lering ende vermaeck. Verder ben ik mijn dank verschuldigd aan PHaio met wie ik een hoop lol gehad heb en bij wie ik een aantal gevleugelde woorden heb mogen introduceren. Vooral mijn (ex)kamergenoten Dick de Ridder, David Tax en Alexander Ypma wil ik bedanken voor de moeite die ze zich getroost hebben mijn proefschrift in de eerste stadia door te lezen. Het zal geen gemakkelijke job geweest zijn, maar ach ook zij zullen ooit een boekje moeten schrijven. Ook dank aan Michiel de Bakker die zich met ziel en zaligheid (tenminste dat neem ik aan) ingezet heeft voor de cover van mijn proefschrift, hulde. Zonder geluk vaart niemand wel, daarom ben ik blij dat ik “research-rots-in-de-branding” Bob Duin als inspirator heb gehad. Van hem heb ik enorm veel geleerd op het gebied van patroonherkenning en onderzoek.

Verder wil ik mijn familie bedanken die op de achtergrond altijd achter mij heeft gestaan. Als laatste wil ik Barbara bedanken die regelmatig mijn afwezigheid heeft moeten ontberen als ik op conferentie of aan een PHaio uitje bezig was en zij de nazorg moest doen. Het was weliswaar niet zo erg als bij mijn afstuderen maar toch ... Bedankt voor je geduld.

Nu rest mij alleen nog te zeggen: ”Het ga u goed”.

Curriculum Vitae

Aarnoud Hoekstra was born on september 6, 1969 in Sneek (Frisia) and went to secondary school in Deventer. After finishing his VWO education in 1987, he went to the university of Twente (Universiteit voor technische en maatschappij wetenschappen Twente) to study computer science. In 1993 he obtained his MSc. in computer science on the subject of the use of neural networks in natural language recognition ("Sequence recognition using the Kohonen feature map").

Later that year, august 1993, he joined the pattern recognition group to perform his PhD. research on neural networks and pattern recognition, which has been finalised in this thesis "Generalisation in Feed Forward Neural Classifiers".