Methods for estimating and improving the stability of neural network classifiers

K.W. Rumpff

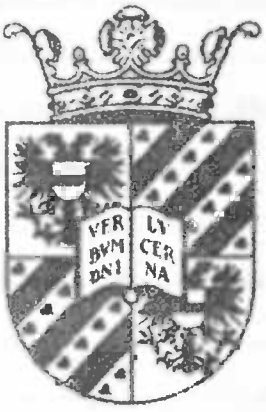
Computing Science

RuG

Methods for estimating and improving the stability of neural network classifiers

by
K.W. Rumpff

under supervision of
Dr. Ir. J.A.G. Nijhuis



Rijksuniversiteit Groningen

Department of Computing Science
Rijksuniversiteit Groningen
Groningen, The Netherlands
January 2000

A Thesis in fulfillment of the requirements for the degree of Master of Science at the Rijksuniversiteit Groningen.

Index

Abstract

One very undesirable effect that quite often occurs when training neural network classifiers, is that the same classifier does not seem to attain the same performance when trained several times, even when (pretty much) the same training data is used. The larger the variation in performance, the less stable (or robust) the classifier seems to be. The stability of a (neural network) classifier can therefore be defined as the probability that the classification result is changed as a result of some small disturbance of the classifier. Such a small disturbance of the classifier can be due to a slightly different training set or to another initial weight setting.

In general, it can be said that the stability of a neural network classifier depends on:

- The structure/architecture (number of hidden neurons, number of hidden layers, etc.);
- The scheme of training (including initial weight setting, order of presentation of training patterns each epoch, but also learning rate, momentum and number of training epochs);
- The size and composition of the training set.

Our primary aim is to derive a method (or methods) that can be used to tell us what kind of stability problems any given classifier might have on any given data set. We are especially interested in whether such methods can also work if only a limited amount of data is available. This, since we assume, that both the size of the training set and the size of the test set will have their (negative) impacts on any possible test outcome. Next we would of course like to now whether (and if so, how) any stability problems that might arise, can be avoided.

In general the stability of a neural network classifier can be estimated by training a number of classifiers and determine the (for instance) the standard deviation over the attained performances on a test set; the higher the standard deviation, the more unstable the classifier seems to be.

But not every set that can be used for testing gives the same result. If the test set is to small in size, or if (part of) the test set is also used to train the classifier, then the results are not representative for the classifiers 'real' performance or 'real' stability. As a result, if the amount of available data is too limited, it is simply impossible to tell whether any results that might indicate an unstable classifier are due to the limited size of the training set or the limited size of the test set. Increasing the amount data has a positive effect on the estimated stability. Using more data for training actually increases the classifiers stability (usually) whereas testing with more data, of course, only gives a more 'stable' estimate. In addition, some amount of instability can be due to the choice of initial setting of the classifier. But as it is usually infeasible to choose the best settings at forehand, there is not much that can be done about this.

Even if we have used enough data and our classifier looks pretty stable we still might have a stability problem. Namely, our classifier might end up working on data with different class probabilities than the probabilities in our training and test sets. If then, this classifier happens to be sensitive to variations in the class ratios then it will most likely perform (far) below its performance, as we estimated it. We have designed a procedure to check in advance whether this might lead to any unexpected problems. We have done experiments on some, artificially

generated data sets and according to our results, this problem does not occur if there is no overlap in our data sets and it appeared to cause much more problems on neural network classifiers with more than one hidden layer. But, besides choosing a proper network architecture there seems to not much that can be done in case of serious class sensitivity problems, other than making sure that the class ratios in the training and test sets match the 'real world' class ratios as good as possible.

Samenvatting

Een nogal ongewenst effect dat regelmatig optreedt tijdens het trainen van neurale netwerk classifiers is dat dezelfde classifier niet elke keer nagenoeg dezelfde performance haalt. Zelfs niet als vrijwel dezelfde data voor het trainen wordt gebruikt. Hoe groter de spreiding in performance, hoe minder stabiel (of robuust) de classifier lijkt te zijn. De stabiliteit van een (neurale netwerk) classifier kan dan ook gedefinieerd worden als de kans dat een classificatieresultaat wijzigt als gevolg van een kleine verstoring van de classifier. Zo een kleine verstoring van de classifier kan het gevolg zijn van een iets gewijzigde trainset of een net iets anders gekozen initiële gewichtsinstelling.

In het algemeen kan gesteld worden dat de stabiliteit van een neurale netwerk classifier afhankelijk is van:

- De structuur/architectuur (aantal 'hidden' neuronen, aantal 'hidden' lagen, enz.);
- Het trainschema (onder andere de initiële gewichtsinstelling de volgorde waarin de patronen per leerslag worden aangeboden, maar ook de 'learning rate', 'momentum' en het aantal leerslagen).
- De grootte en de samenstelling van de trainset.

Het primaire doel is het vinden van een methode (of methodes) die gebruikt kan worden om aan te geven wat voor soort stabiliteitproblemen een willekeurige classifier kan geven op een gegeven dataset. We zijn vooral geinteresseerd in of dergelijke methodes ook werken als slechts een beperkte hoeveelheid data beschikbaar is. Dit, omdat we aannemen dat zowel de beperkte grootte van de trainset als van de testset een (negatieve) invloed zullen hebben op elk mogelijk testresultaat. Daarnaast zouden we uiteraard graag willen weten of er ook nog iets tegen stabiliteitproblemen te ondernemen valt.

Normaal gesproken is de methode om de stabiliteit van een classifier in te schatten doe volgende. Train een aantal netwerken en bepaal (bijvoorbeeld) de standaard deviatie over de behaalde performances; hoe hoger deze standaard deviatie hoe minder stabiel de classifier lijkt te zijn.

Maar niet elke dataset die kan worden gebruikt om de netwerken te testen geeft hetzelfde resultaat. Als de testset te klein is of als (een gedeelte van) de testset ook gebruikt is om de netwerken te trainen dan zullen de resultaten niet representatief zijn voor 'werkelijke' performance en de 'werkelijke' stabiliteit van de classifier. Het gevolg hiervan is dat als een beperkte hoeveelheid data beschikbaar is, het simpelweg onmogelijk is om vast te stellen of resultaten die duiden op instabiliteit het gevolg zijn van de beperkte trainset of van de beperkte testset. De (geschatte) stabiliteit kan worden vergroot door meer data te gebruiken. Het gebruik van een grotere trainset verbetert in de regel werkelijk de stabiliteit van de classifier, terwijl een grote testset (alleen) een 'stabielere' schatting geeft. Daarnaast is een gedeelte van de gemeten instabiliteit het gevolg van de keuze van initiële instellingen van de classifier. Maar aangezien het in de regel niet doenlijk is om de meest geschikte settings op voorhand te kiezen, kan hieraan weinig verholpen worden.

Zelfs als we genoeg data hebben gebruikt en onze classifier lijkt behoorlijk stabiel te zijn, dan is het nog steeds mogelijk dat we een stabiliteitprobleem hebben. Het namelijk mogelijk dat deze classifier zal worden toegepast op data met andere klasseverhoudingen dan de verhoudingen waarop het getraind en getest is. Als in dat geval, deze classifier nogal gevoelig blijkt te zijn voor veranderingen in de klasse-ratios dan zal het waarschijnlijk (ver) beneden zijn ingeschatte performance presteren. Hiervoor hebeen we een procedure ontworpen die gebruikt kan worden om vooraf na te gaan of een dergelijke situatie kan leiden tot onverwachte problemen. Uit de resultaten van experimenten op een aantal, artificieel gegenereerde datasets blijkt dat dit probleem zich in elk geval niet voordoet las de klassen in de dataset geen overlap vertonen. Daarnaast blijkt dat netwerken met meerdere 'hidden' lagen meer problemen ondervinden dan netwerken met een enkele laag 'hidden' neuronen. Maar naast het kiezen van de meest geschikte netwerkarchitectuur lijkt het erop alsof aan dit probleem verder weinig gedaan kan worden. Het beste kan er dus naar gestreefd worden om de klasseverhoudingen inde train- en testset zoveel mogelijk overeen te laten komen met de verhoudingen zoals ze in de situatie liggen waarvoor deze classifier uiteindelijk bedoeld is.

Chapter 1 Introduction

In this thesis we will concentrate a problem that is commonly encountered when using neural networks for classification problems, namely its lack of *stability* under certain circumstances. But first we will first give a short introduction in the field of classfication with neural networks followed by a description of stability in relation with classifiers and of the problems related to a lack of stability. For a more extensive introduction in classification and neural networks see Ripley (1996) and Haykin (1999).

1.1 Classification

Classification, or *pattern classification*, is the task of (correctly) labeling (classifying) the examples that are presented to it. The classifier is allowed to label each example as to belong to any of a fixed number of categories (classes), or it allowed to tell that it is too hard to classify the given example. Pattern classification can be (and is) applied in wide range of areas including:

- diagnosing diseases,
- identifying car number plates,
- reading hand-written symbols,
- detecting shoals of fish by sonar,
- classifying galaxies by shape.

Obviously, when looking at the list of classification tasks above, before we can let any computer algorithm do the classification task we need to able to feed it with the right *features*, that is the right *measurements* have to be made. And often we have to go even further, that is we have to do the right *preprocessing*, that is translating the original measurements into additional 'indirect measurements' to ease the task for the computer. A simple example. If we want to classfy point on a 2D surface we could take the x- and y-coordinate of each point, but we can also (in addition or instead) translate these values in the corresponding radius and angle combination (with respect to a choosen origin). (Applying this transformation to the concentric database as described in Appendix B, it would have certainly given us less trouble (see Chapter 4). But since it is not always this easy to spot better features, we have choosen to use the database as it was given.)

1.2 Neural networks

When we are talking about neural networks we actually mean *artificial* neural networks. There are several kinds of artificial neural networks that can be used for various tasks but we will restrict ourselves to one of the most common networks, namely the *multi-layer perceptron*. To understand what such a neural network is and does it is best to first understand how a neuron works. A neuron is an information-processing unit that is fundamental to the operation of a neural network. The following three basic elements of the neuron model may be identified:

- A set of *synapses* or *connecting links*, each of which is characterized by a *weight* of its own. Specifically, a signal x_j at the input of synapse j connected to neuron k is multiplied by the synaptic weight w_{kj}.
- An *adder* for summing the input signals, weighted by the respective synapses of the neuron; the operation described here constitute a *linear combiner*.
- An *activation function* for limiting the amplitude of the output of the neuron. Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval [0,1] or alternatively [-1,1].

In Figure 1.1 the model of a neuron is shown. In mathematical terms, we may describe a neuron k by writing the following pair of equations:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \tag{1.1}$$

and

$$y_k = \varphi(u_k - \theta_k) \tag{1.2}$$

where $x_1, x_2, ..., x_p$ are the input signals; $w_{k1}, w_{k2}, ..., w_{kp}$ are the synaptic weights of neuron k; u_k is the *linear combiner output*; θ_k is the *threshold*; $\varphi(.)$ is the *activation function*; and y_k is the output of the neuron. The use of the threshold θ_k has the effect of applying an *affine transformation* to the output u_k of the linear combiner in the model of Figure 1.1, as shown by:

$$v_k = u_k - \theta_k \tag{1.3}$$

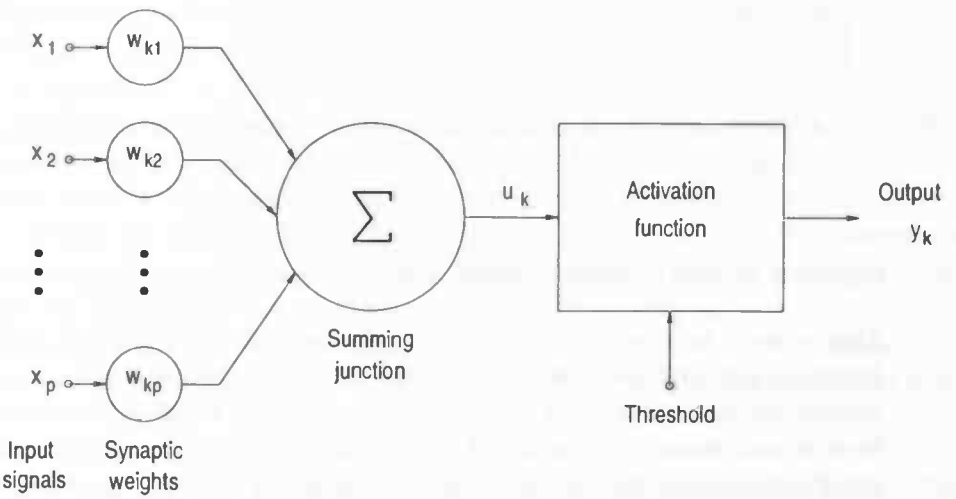


Figure 1.1: Nonlinear model of a neuron.

For the activation function we may identify three different functions (examples of the latter two are depicted in Figure 1.2):

- *Threshold function*. This is the most simple function and is defined as:

$$\varphi(v)=\begin{cases}1 & \text{if } v\geq 0\\ 0 & \text{if } v<0\end{cases} \tag{1.4}$$

- *Piecewise-linear function*. This function can be defined as follows

$$\varphi(v)=\begin{cases}1 & \text{if } v\geq \frac{1}{2}\\ v & \text{if } -\frac{1}{2}>v>\\ 0 & \text{if } v\leq -\frac{1}{2}\end{cases} \tag{1.5}$$

  where the amplification factor inside the linear region of operation us assumed to be unity.

- *Sigmoid function*. The sigmoid function is by far the most common form of activation function used in the construction of artificial neural networks. It is defined as a strictly increasing function that exhibits smoothness and asymptotic properties. An example of the sigmoid is the logistic function, defined by

$$\varphi(v)=\frac{1}{1+\exp(-av)} \tag{1.6}$$

  where a is the *slope parameter* of the sigmoid function.

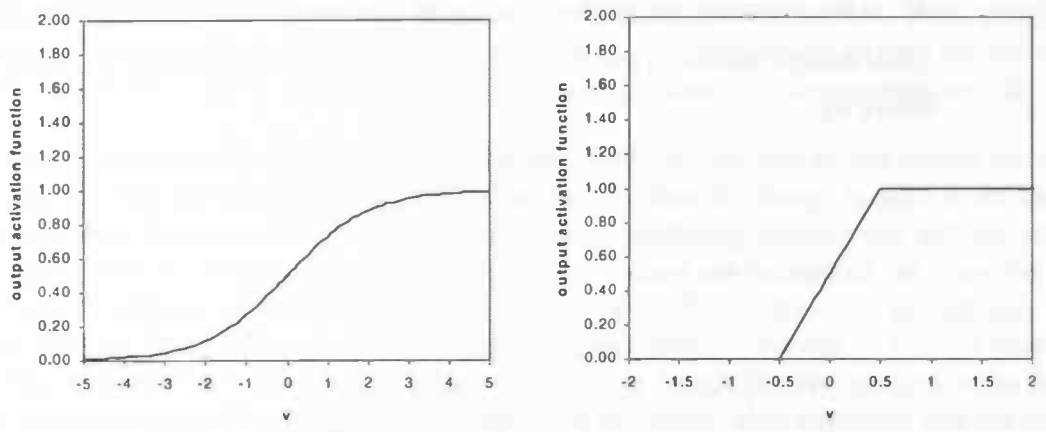


Figure 1.2: Examples of activation functions. (left) Sigmoid function. (right) Piecewise-linear function

This neuron, as described above, with adjustable synaptic weights and threshold is also known as a *perceptron*. If we now combine several of these perceptrons to construct a multi-layered network as depicted in Figure 1.3 (each gray circle depicts a perceptron) we then have an architecture of an artificial neural network also known as the *multi-layer perceptron* (MLP). This under the restriction that the activation functions used in each perceptron, are smooth that is, differentiable everywhere (like for instance, the sigmoid function is). Note that the network shown here is fully connected, that is, the output of each neuron in one layer is connected to the input of each neuron in the next layer.

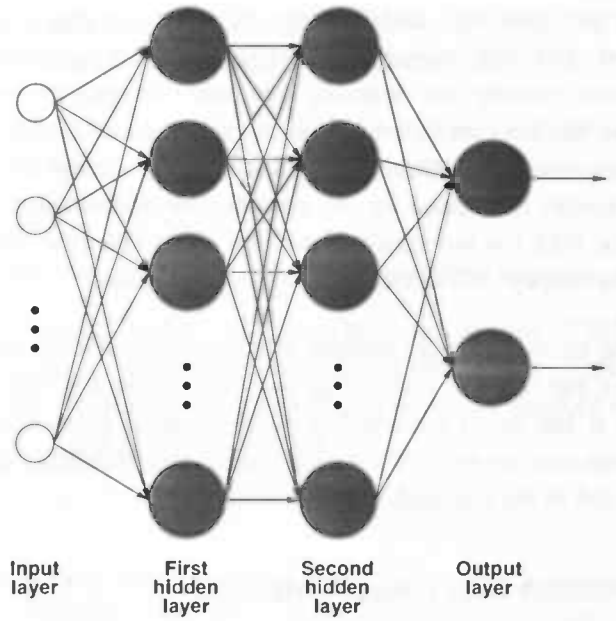


Figure 1.3: Architectural graph of a multilayer perceptron with two hidden layers.

It is proven (by Cybenko, see Haykin, 1999) that a MLP with only a single hidden layer (with a sufficient number of neurons) is capable of uniformly appoximating any continuous function with support in a unit hypercube (due to the activation functions an MLP can only produce outputs within a limited range, for instance [0,1]). In practice things are a bit more complicated. First, there is the choice of the number of hidden neurons, for which is there is no (simple) rule (but rules of thumb). To complicate things a bit more, since in practice we can only choose a limited number of hidden neurons it might very well be possible that choosing less neurons, but arranging them in several hidden layers is actually a better choice than simply using a single hidden layer. Then there is the setting of the synaptic weights and thresholds. It for any practical problem not possible to select the proper settings in advance. We will therefore instead initialize the weights (including the threshold, which can also be considered as a special weight) to small random values and adjust the weights by means of a *learning algorithm*. A highly popular learning algorithm is called the *error back-propagation algorithm*. For an in depth explanation of this algorithm see Haykin (1999). We will limit our explanation to the consequences of the use of this algorithm. First of all it requires a set of training samples, that is a set of input vectors and corresponding desired output vectors. The network is then to be supplied with each of these train samples in turn and a corresponding error vector (desired outputs - outputs) will be computed. This error measure will then be back-propagated through the network (that is, from output to input) and adjustments to each of the weights will be made. This process is then to be repeated until, for instance, the sum of squared errors made on these training samples has reached an acceptable value.

When looking at the theory, it would seem like the only acceptable value is zero. This is in practice not a realistic goal for several reasons. First of all, the network at hand may not be large enough to fully implement the function at hand. Second, the training set might be rather small and/or it might even be 'polluted' by some erroneous examples. In this case you do not even want the network to fully learn this training set. And last, but not least, applying the error back-propagation algorithm is no guarantee to find the optimum solution at all. The error back-propagation algorithm does its job by trying to find the lowest point in a so called error surface by calculating (an approximation of) the gradient at each point. But when the

gradient becomes zero and the network stops learning it might have only reached a local minimum. For this reason the back-propagation algorithm is usually controlled by two parameters, namely the learning rate and the momentum parameter. The learning rate parameter has only an direct influence on the rate of change of each weight. The momentum parameter makes sure that when calculating the change of each weight the previous change of this weight multiplied by the momentum is also taken in account. By choosing proper values for both the learning rate and the momentum the back-propagation algorithm should be able to simply 'roll through' minor local minima.

When the network is fully trained, it is hoped that the network has not only learned the given examples, but also the underlying function so that it is able to give a proper output for input samples it has never seen before (this is called generalization). This is unfortunately not always the case, as we will see in the paragraph **Stability related problems** further on in this chapter and in the remainder of this thesis.

1.3 Classification with neural networks

Neural networks can not only be used for function approximation but also for pattern classification. For this purpose we simply give our network as many outputs as there are categories (classes) so that each output neuron corresponds to one category. If we are now talking about classification problems without overlap (that is, each possible input vector can only belong to one class), then it must be possible to train a network in such a way that for each input it gives an output vector that consists of all zeros and a single one. The output with value one then corresponds to the class to which the patterns belongs. Well, actually, since a sigmoid function is never able to reach 0 or 1 it is customary use to train the network with values $0+\varepsilon$ and $1-\varepsilon$ instead, where ε is a small value, for instance equal to 0.05.

When we are talking about classification problem s with overlap (that is, one pattern belongs to class A and another, but equal pattern belongs to class B) things become a bit more complicated. The network now, has to learn to map conflicting patterns and will therefore never be able to classify all patterns correctly. Since we cannot expect the classifier to classify all patterns correctly, we settle for a correct prediction of the class probabilities for each pattern. That is, if a given pattern has a higher chance to belong to class A than to belong to class B then the output corresponding to class A should show a higher value than the output corresponding to class B. It is therefore, in general, very important that the class ratios (probabilities) in the training set not only correspond to the class ratios in the test set but also to the 'real world' situation to which we want to apply this trained classifier.

The performance of a neural classifier is usually measured by the percentage of patterns from a test set that is correctly classified, usually by taking the class corresponding to the output with the highest value as the winning class. (The error rate is equal to the percentage of incorrectly classified patterns.) Note that even though a classifier may produce the correct class probabilities for each input patterns, it will never be able to score a performance rate of 100% if the above described measure is used.

1.4 Stability related problems

One very undesirable effect that quite often occurs when training neural network classifiers, is that the same classifier does not seem to attain the same performance when trained several times, even when (pretty much) the same training data is used. The larger the variation in

performance, the less stable (or robust) the classifier seems to be. The stability of a (neural network) classifier can therefore be defined as (Hoekstra et al, 1997):

The probability that the classification result is changed as a result of some small disturbance of the classifier. Such a small distturbance of the classifier can be due to a slightly different training set or to another initial weight setting.

In general, it can be said that the stability of a neural classifier depends on:

- The structure/architecture (number of hidden neurons, number of hidden layers, etc.);
- The scheme of training (including initial weight setting, order of presentation of training patterns each epoch, but also learning rate, momentum and number of training epochs);
- The size and composition of the training set.

1.5 This thesis

In this report we will merely concentrate on the effects of the size and composition of the training set on the stability and more or less ignore the effects of the choice of structure/architecture and scheme of training. Of course some suitable architecture must be chosen and some appropriate scheme of training must be used, but that will be no major point for attention. Further more, we assume that, the larger the training set is, the more stable the resulting classifier will be (and the better it's average performance). If only a limited amount of data is available, matters are complicated by the fact that, though the size and composition of the test set have no influence on the stability of the classifier, they will (probably) have an influence on the classification result.

Our first aim is (thus) to derive a method, to estimate how stable a classifier can be obtained if only a limited data set is provided. We will assume that class ratios in the given data sets correspond to the a posteriory class probabilities (that is, the probabilities as they appear in the 'real world'). This can easily be mimicked by using the same class ratios in our test sets as in the training sets. Our hope is that the results of this research will make it possible to make an educated guess (given only a limited amount of data) about the amount of data that is needed to build and test a neural classifier that attains a certain, guaranteed performance.

Of couse, it is not always possible, to be so sure about the a priory class probabilities. And, the class ratios in a given data set may not always correspond to both the a priory and the a posteriory class probabilities. Thus it might be possible that one ends up with a classifier, that is seemingly stable, but when applied to 'real world' data it shows a considerable lower performance as seemed to be expected. Therefore we would also like to have a procedure that can tell us more about any stability problems that might arise when applying a given data set with incorrect class probabilities (with respect to the a posteriory probabilities).

In Chapter 2, we will give a summary of some theory about how to estimate a classifiers stability and about the influences that the training and test sets can have on this estimate.

In the literature several methods, that are supposed to have a stabilizing influence on neural classifiers, are described. These methods seem to concentrate mostly on the problem called overfitting, meaning that the classifier is able to learn some of the noise in the training data or is even able to more or less memorize the training set. This problem can (amongst others) arise when the classifier has too large a complexity (with respect to the number of training patterns). Though overfitting seems to be a serious problem that enjoys a lot of attention in

the literature, no real signs of it showed during any of the experiments that were carried out in the name of this project. A summary of these methods is given anyway in Chapter 3.

Then, in Chapter 4, we will derive our own procedure for estimating the stability. With the help of this procedure we will also see whether any of the theory, given in Chapter 2 holds in these practical situations. And whether on the basis of results on experiments with only a small amount of data predictions can be made about the attainable performance and/or stability when a larger amount of data is available.

So far, we have taken the proportions in the given data sets as a priory chances and therefore have kept them constant (in none of articles we have referred to so far, anything else was tried). In Chapter 5, however, we are interested in the influence of varying class ratios on the performance of the resulting classifier. This time we will also take a look at classifiers with two hidden layers and see how well they perform (in terms of stability) compared the single-hidden-layer networks we have been using so far.

And finally, in Chapter 6 a summarization of the conclusions we have come to will be given.

Chapter 2 Other research on stability

Most of the theory about stability is about (the prevention of) overfitting (see Chapter 3). But there is also some theory about how to estimate a classifiers stability. And about how sizes and compositions of training and test sets can have an influence on this estimate. Some (short) descriptions of these theories will be given in the following paragraphs.

2.1 A procedure for estimating the stability

One suggested procedure to estimate the stability of a classifier (Hoekstra et al, 1997) works as follows:

1. First the entire training set is used to compute a classifier S_0.

2. Next a large number of classifiers S_i (i = 1,2,...,n) is computed using one of the following procedures:
   - For each classifier leave one sample out of the training set;
   - Bootstrap the training set, that is, sample with withdrawal;
   - Use a different initial weight setting for each classifier.

3. Compute the average different classification between S_0 and the classifiers S_i, averaged over the n classifiers S_i and over the test set. For this test set one of the following sets might be used:
   - The original training set;
   - A truly independent test set.

In the last step of this procedure, simply the number of different estimated classification labels is counted. The true labels are not used. The result is number that lies between 0 and 1. The more this number approaches zero, the more stable the classifier seems to be, while a resulting number close to one indicates that the classifier is far from stable.

This procedure has a couple of drawbacks. In the first place, this procedure makes use of an appointed average classifier, S_0. Especially with rather unstable classifiers, this appointed average will show a lot of variation when trained several times on the same training set.

In the second place, Hoekstra et al. state that when estimating the stability, the test set can be the original training set as well as a truly independent data set. They do not explain why the choice of the test set does not affect the resulting estimate, neither do they make any statements about the size of test set that is required to make a reliable estimate of the stability.

2.2 Effects of using too small a train and/or test set

Some interesting statements about the use of a limited data set (which will lead to limited test and train sets) are made by Fukunaga (1990). Though Fukunaga has tried to keep the theory as general as possible, his derivations and illustrations are mostly based on the use of linear and quadratic classifiers and no mention is made about neural network classifiers.

He comes to the following conclusions:

- The *bias* of the classification error comes entirely from the finite design (training) set;
- The *variance* comes predominantly from the finite test set.

The bias, in this context, can be seen as the difference in classification error (or performance) with respect to the optimal (minimal) error that can possibly be obtained on any particular classification problem. The variance is calculated from the differences in estimated errors (or performances) .

2.3 What test set size gives a good error rate estimate

In order to make a good estimate of the error rate (and thereby of the performance) of a classifier, a large test set is needed. The question remains what size of test set gives a reliable error rate estimate? For a more elaborate description of the following theory see see Duda and Hart (1973) and Guyon et al (1998).

If the true but unknown error rate of the classifier is p, and if k of the n independent, randomly drawn test samples are misclassified, then k has the binominal distribution:

$$P(k) = \binom{n}{k} p^k (1-p)^{n-k} \tag{2.1}$$

Note, however, that for this assumption to be satisfied, the samples must be chosen randomly, which, in multi-class problems, might lead to some classes not being represented at al. To circumvent this problem, it is common practice to the number of test samples in each class correspond, at least roughly, to the a priory probabilities. Though this improves the estimate of the error rate, it complicates an exact analysis.

For a test set size of n examples, the following is an estimate of p :

$$\hat{p} = \frac{k}{n} \tag{2.2}$$

Since we would like to know how reliable this estimate is, we are also interested in confidence intervals. With a certain confidence $(1-\alpha), 0 \le \alpha \le 1$, we want the expected value of the error rate p to be either within a certain range:

$$\hat{p} - \varepsilon(n,\alpha) < p < \hat{p} + \varepsilon(n,\alpha) \tag{2.3}$$

(two-sided risk)., or simply not to exceed a certain value:

$$p < \bar{p} + \varepsilon(n,\alpha) \tag{2.4}$$

(one-sided risk). Since it is not of our concern if the value of the expected error rate is better than what we estimate, only the one-sided risk will be used.

The random variable, of which $\bar{p} + \varepsilon(n,\alpha)$ is a realization, is a guaranteed estimate of the mean. We are guaranteed, with risk α of being wrong, that the mean does not exceed $\bar{p} + \varepsilon(n,\alpha)$.

For a binomial distribution Guyon et al (1998) derive the following, pessimistic bound, that asserts that with probability $(1-\alpha)$:

$$p - \hat{p} < \sqrt{-2\ln\alpha}\sqrt{\frac{p}{n}} \tag{2.5}$$

If we introduce the variable β such that $\varepsilon(n,\alpha) = \beta p$ then the number of test samples needed to satisfy Eq. (2.5) is:

$$n = \frac{-2\ln\alpha}{\beta^2 p} \tag{2.6}$$

As can be seen, the number of test samples needed is independent of the number of classes. And again, Eq. (2.5) and Eq. (2.6) are only derived for the case in which the test samples are chosen randomly, which can result in some classes not being represented at all.

There is still one problem left: we need to know the true error rate p in order to calculate n. We are left no choice, but to use the largest test set available to us (which is probably much too small) and use this set to make an estimate of p according to Eq. (2.2) and then use $\hat{p}$ instead of the true error rate.

2.4 Conclusions

Though we have only found one procedure for estimating the stability of a classifier, it is customary use when training a classifier, to actually train a (small) number of classifiers and check whether their performances do not vary too much. How else can one explain the amount of attention that stabilizing techniques enjoy (see Chapter 3 for more information about these techniques). Only, usually not much mention is made about the exact procedure that is followed to determine whether the classifier at hand is stable or not. According to Heokstra et al. (1997) this is more or less justified, since they state that it does not really matter what composition of the test set is used. Neither do they make clear whether the size of the test set can have any influence on the estimate of the stability. However, these assumptions do not agree with the theories of both Fukunaga and Guyon et al. According to both their theories the size of the test set does matter while demanding this test set to be independent of the training set (Apparently they do not have much confidence in results obtained by using a single data set both for training and for testing). Anyway, we will return to most of these assumptions and theories in Chapter 4 where we will discuss their validity in practice.

Chapter 3 Stabilizing techniques

There are several techniques that are known to have a stabilizing effect on neural network classifiers. In the following paragraphs a summarization of six well-known techniques will be given. These methods have in common that they aim to reduce the space in which is searched for an optimal classifier (by reducing the number of (local) minima in the error surface) and thereby reducing the number of different, resulting classifiers. However, applying these methods does not always have the disired effect: it is very well possible that the global minimum is also affected. As a result, one might end up with a classifier that *is* more stable, but has a far worse average performance. Therefore, if the effects of any of the following teechniques on the stability is not noticable, it might be best to not apply it at all. It also should be noted that the last two techniques, pruning and bagging, differ from the first four techniques since they do not actually stabilize a given classifier. Instead, pruning reduces the architecture of a given (unstable) classifier and bagging combines several (unstable) classifiers in order to get a stable result. More information about these methods can be found, for instance, in Haykin (1999), Ripley (1996) and on the following ftp-site maintained by W.S. Sarle: "ftp://ftp.sas.com/pub/neural/FAQ.html".

3.1 Training with noise (jitter)

Training with noise can take place in at least two different ways. One way of training with noise consists of deliberately adding artificial noise (jitter) to the inputs of the network during training. That is, each time an input pattern is presented to the network (during training) a different amount of noise is added to it while the target kept unchanged. The basic idea behind this method is that if two inputs are similar, the desired outputs will usually be the same. Of course this means that the amount of noise that is added should not be too large for this will not result in similar inputs. The amount of noise should also not be too small or it will have no effect at all. Usually a normal distributed noise with a zero mean is used. When the proper amount of noise is chosen this method seems like a convenient way to enlarge the training set and therefore to improve the stability of the resulting classifier.

Another way of using noise to improve the stability of the resulting classifier is to add the noise to the hidden units, instead of to the inputs. In this case a different amount of noise is added to the sum of inputs of a hidden neuron, before the signal enters the activation function. Tsukunda et al (1995) describe that neural classifiers that are trained in this way obtain good generalization for any number of hidden neurons (any complexity).

3.2 Early stopping

One of the most popular methods to prevent overfitting is early stopping. It starts with an oversized network, but it is not trained until the smallest test error has been reached. Instead a validation set (a small test set during training) is used to determine when to stop training. By this way the network is not given enough time to learn any of the noise that occurs in the training set, or to simply store all the given training samples. The resulting network is supposed to have learned a smooth approximation of the signal in the training set. The method proceeds as follows (Sarle, 1995):

- divide the available data (excluding the test data) into two sets: a training and a validation set;
- use a large number of hidden neurons (an oversized network);
- use small random initial weights;
- use a slow learning rate;
- compute the error on the validation set periodically during training;
- stop training when the validation error "starts to go up".

The advantages of early stopping are that it is fast and that it has no special parameters that need to be tuned; it only requires the decision of what size the validation set should be. But there are also some unresolved practical issues in early stopping:

- What sizes should the training and validation sets be?
- How should the data be split into the training and validation sets; randomly or by some systematic algorithm?
- How can one tell that the validation error "starts to go up"? It might go up and down several times during training. The safest approach is to train to convergence while saving the network periodically. When training is finished the network with the smallest error on the validation set can be chosen.

3.3 Weight decay

Weight decay is a way to constrain a large network, and thus decrease its complexity, by limiting the growth of the weights. It aims to prevent the weights from growing too large unless it is really necessary. This can be realized by minimizing the total risk expressed as:

$$R(\mathbf{w}) = E_s(\mathbf{w}) + \lambda E_c(\mathbf{w} \tag{3.1}$$

The first term, $E_s(\mathbf{w})$, is the standard performance measure; in back-propagation learning it is usually the sum of squared errors. The second term, $E_c(\mathbf{w})$, is the complexity penalty and λ is a regularization parameter, which represents the relative importance of the complexity penalty term with respect to the standard performance measure term. $\mathbf{w}$ is a vector containing all free parameters in the network; the weight vector.

The complexity penalty term can be defined in several ways. In the standard weight decay procedure the penalty term is defined as the squared norm of the weight vector:

$$\begin{aligned} E_c(\mathbf{w}) &= \tfrac{1}{2}\|\mathbf{w}\|^2 \\ &= \tfrac{1}{2}\sum_{i \in W} w_i^2 \end{aligned} \tag{3.2}$$

where W is the set containing all the synaptic weights of the network. If gradient descend is used for learning, the weight update becomes:

$$\Delta w_i = -\eta\left(\frac{\partial E_s}{\partial w_i} + \lambda w_i\right) \tag{3.3}$$

where η is the learning rate parameter. It can easily be seen from Eq. (3.3) that all weights are treated equally and are forced to take values close to zero. Only weights that have a large influence on network are able to resist this force. Weights that have little or no influence on the model, the so-called excess weights, will not be able to resist and will therefore take values close to zero. Without any form of weight decay these excess weights might take arbitrary values or cause the network to overfit the train data in order to produce a slightly smaller training error.

In an extension of the weight decay procedure, which allows some weight in the network to assume values that are larger than with the standard weight decay procedure, the penalty term is defined as follows:

$$E_c(\mathbf{w}) = \sum_i \frac{(w_i / w_0)^2}{1 + (w_i / w_0)} \tag{3.4}$$

where w_0 is a preassigned parameter. When $|w_i| << w_0$, the complexity penalty for that weight approaches zero. When, on the other hand, $|w_i| >> w_0$, the complexity penalty for that weight approaches one. A plot of this complexity penalty, with $w_0 = 1$, is given in Figure 3.1. Note that for large w_0 Eq. (3.4) reduces to Eq. (3.2) except for a scaling factor.

Though the weight decay procedure does encourage excess weights to assume values close to zero and thereby improves the stability of the classifier, it can also have a less desired side effect. It namely assumes that the prior distribution in weight space is centered at the origin, in other words, at forehand it assumes that the best suitable network is a network with all weights set equal to zero (which is, by the way, a very stable network). Thus, the larger the regularization parameter λ is, the smaller the (relative) total risk will be around the origin. As a result the use of a weight decay procedure can even change the position of a global minimum in the error-surface. This is depicted in Figure 3.1 (G. Thimm and E. Fiesler, 1997), with w_0 and λ both set equal to one: the standard error performance measure alone has a global minimum at $\mathbf{w} \approx 2$ and a local minimum at $\mathbf{w} \approx 0.3$ while, due to the complexity penalty, the total risk has a global minimum at $\mathbf{w} \approx 0.1$. So care should be taken when using a large value for λ.

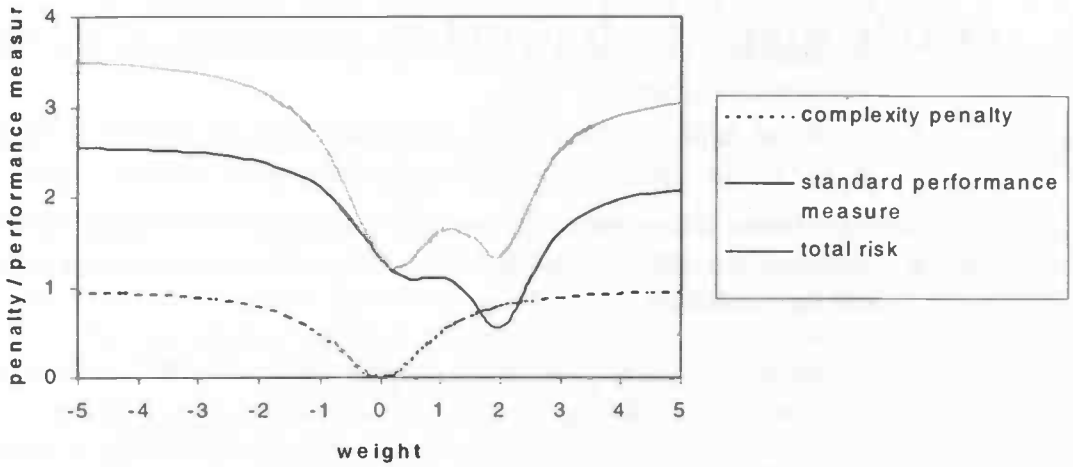


Figure 3.1: The penalty term may change the location of global minima.

3.4 Initial weight setting

Atiya and Ji (1997) show that the initial weight setting can have an influence on the generalization performance of neural classifiers. If the initial weights are chosen small then there is a tendency that the final weights are small. This effect is similar to having some kind of weight decay.

3.5 Pruning

Pruning methods are a basic approach to find a nearly minimal neural network topology. This minimal network topology is the smallest possible network that performs good generalization without any serious under- or overfitting and should therefore be rather stable. A pruning algorithm starts with a sufficient large, fully connected network and works by ranking the weights and then removing the least important ones. Pruning is typically applied through the following steps:

1. Train a 'sufficiently large', fully connected network;
2. Rank the weights and remove the least important one(s);
3. Retrain the network and return to step 2 if further pruning is desired..

An obvious question is what criterion should be used to stop pruning. A validation set can be used for this problem. After the network is (re)trained till convergence the network is tested on the validation set: when this error "starts to go up" the network might become to small for the problem at hand. Note that this stopping criterion looks very similar to the one used with early stopping and shows the same problems (see section 3.2).

For the ranking of the weights several methods can be used. Next follow some relative simple heuristics as described by G. Thimm and E. Fiesler (1997):

- The simplest heuristic selects the connections with the smallest weights. In addition to the commonly method, which removes only connections, the growth of the error can also

be reduced by adding the connection's mean contribution $\frac{1}{P}\sum_{p=1}^{P} c^p$ (its output c averaged over the train set of size P) to the bias of the neuron receiving input from the removed connections.

- A second heuristic removes the connection with the smallest contribution variance $\sigma_p(c^p)$. The method is therefore called the smallest variance ($min(\sigma)$) method. The mean output of the removed connections is added to the corresponding bias.
- This final heuristic estimates the sensitivity s of the neural network to the removal of a certain weight by:

$$s = \sum_{n=1}^{N} \begin{cases} (\Delta w(n))^2 \dfrac{w(n)}{\eta(w(n) - w(0))} & \text{if } w(n) \neq w(0) \\ 0 & \text{otherwise} \end{cases}$$

where $w(n)$ is the weight in the current training epoch n, $w(0)$ the initial weight and N is the number of training epochs.

There are two other, well-known pruning techniques that need to be mentioned: the *Optimal Brain Damage* (OBD) *procedure* and the *Optimal Brain Surgeon* (OBS) *procedure* (which includes the OBD procedure as a special case). Both procedures are based on a full Hessian matrix and have a high computation complexity. For an elaborate description of these procedures and of the Hessian matrix see (for instance) Haykin (1999).

3.6 Bagging

Bagging is a technique that combines the results of number of classifiers to produce a single classifier. The resulting classifier is generally more stable and more accurate than any of the individual classifiers is (Maclin and Opitz, 1997), while the stability of the individual networks is not influenced. The individual classifiers are all trained on a slightly different training set, by bootstrapping the original training set. An effective way of combining the results of the individual classifiers is by simply averaging them.

3.7 Conclusions

As we have seen, there are lots of stabilizing techniques that can be applied to unstable classifiers. Most of these techniques are especially meant to prevent a classifier from overfitting. Even though overfitting may be a serious problem, in our experiments no (serious) signs of overfitting ever showed. That is the main reason why we have not taken a closer look at any of these techniques. An other reason is that most of these techniques work by confining the space of possible resulting classifiers. One simply cannot tell at forehand whether by this means also the optimal classifier becomes unreachable. And whether any of the classifier that still can be obtained even comes close to this performance. Therefore it may be best to not use any of these restricting methods unless one is really experiencing stability problems that are mostly due to overfitting.

Chapter 4 Stability on limited amounts of data

In this chapter we will derive a procedure for estimating the stability, preferably with the help of only a limited amount of data. This procedure will then be applied to a number of classification problems (three in total) for which only a limited amount of data is available. We will also see whether any of the theory, given in Chapter 2 holds in these practical situations. And whether on the basis of results on experiments with only a small amount of data predictions can be made about the attainable performance and/or stability when a larger amount of data is available. We will carry out all experiments while keeping the given class ratios intact, that is we will use equal class probabilities for both the training and corresponding test sets. This is namely the most common (and intuitive) method of doing experiments (see for instance Duda and Hart, 1973). In this chapter, we will also not include a study of the effects of using other network architectures than a multi-layer perceptron with only a single hidden layer, that is the network we are most accostumed to and which has proven to be fully capable of learning any of the data sets that are to used in the foollowing experements. For more on the effects of varying class ratios on the stability and a comparison of neural network classifiers with only a single and two hidden layers see Chapter 5.

4.1 Determining suitable training schemes

Though we are about to more or less ignore the impacts that a training scheme can have on the stability of the resulting neural network classifier, some suitable scheme of training must be chosen anyway. Since all following experiments will be carried out with the help of three different types of data sets, we will have to determine three, different training schemes.

The general procedure that we have followed to determine the suitable training schemes is the following. First a small number (say three) of networks is trained on a part of the available data for a large number of epochs (say 1000) with 10 hidden neurons and while using 'default' learning rate parameters. The 'default' value for the learning rate is 0.6 and for the momentum 0.2. Next the resulting learning curves are to be given a close look to see how many epochs are required to achieve a more or less stable recognition performance. This procedure can then be repeated for a different number of hidden neurons (say, 20 or 30) to see if this makes any noticeable difference.

Clouds data

The clouds database (see Appendix Appendix A) is a 'difficult' data set in the sense that the two classes it contains severely overlap each other. It is therefore no surprise to see that while 1000 epochs seem to be enough to somewhat stabilize the (average) recognition rate, it still shows quite some variation. A simple remedy is to let the learning rate parameter decrease slowly to somewhere close to zero. This will surely lead to a stable recognition rate without any variation, but the question that remains is, whether this final value will be the average, the minimum or the maximum of the variation it shows. There is no definite answer to this question: the final value of the training set seems to settle on a value above average, but the final value on the validation set can just as well end around its minimum. Using more

than 10 hidden neurons does not seem to make any sense. In Figure 4.1, the architecture that we will use, is depicted.

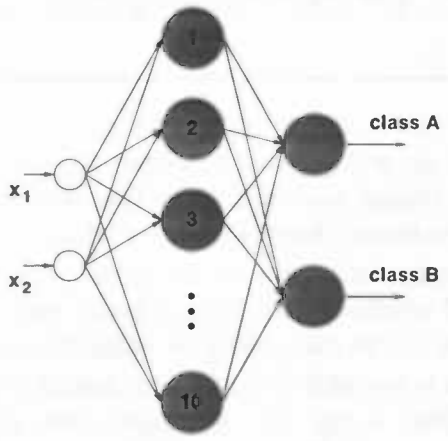


Figure 4.1: The network architecture used for the clouds database.

Table 4-1 shows the final training scheme that is used for the experiments described in the following sections and Figure 4.2 shows a typical learning curve when using this scheme.

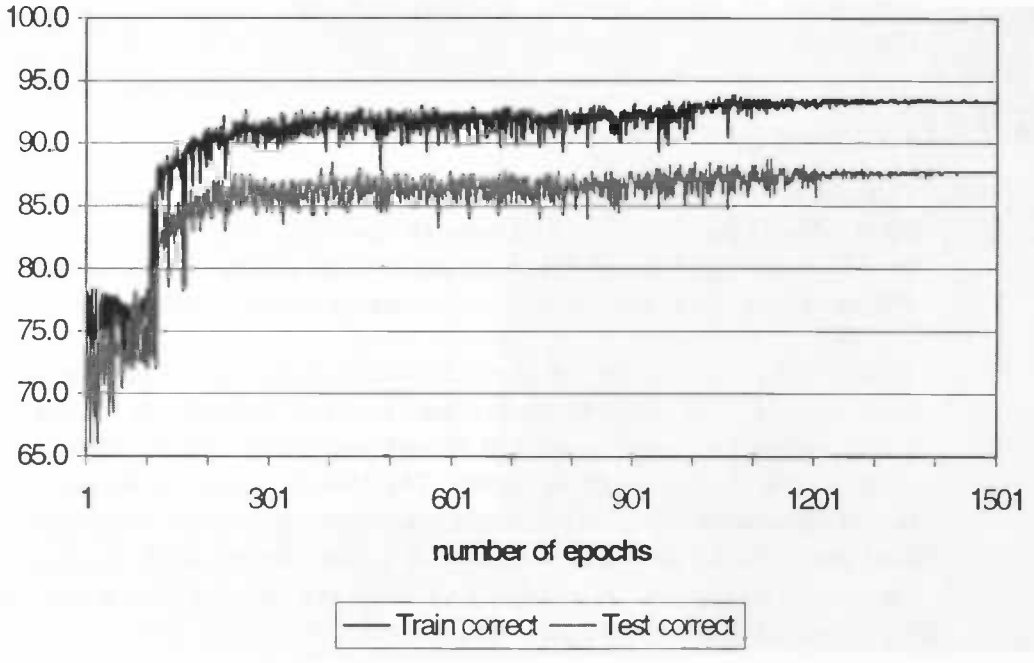


Figure 4.2: Typical learning curves for the clouds data set.

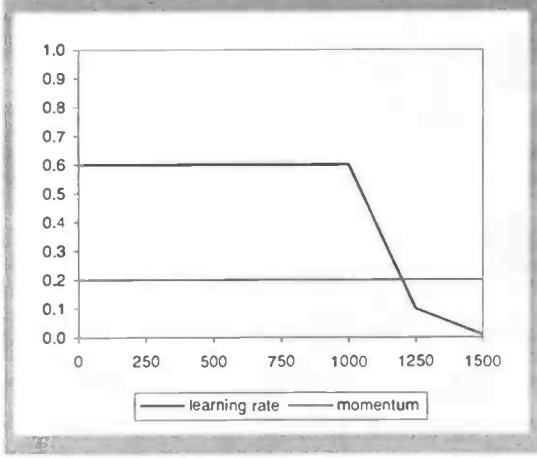
plot	data points		
	epoch:	learning rate:	momentum:
	0	0.6	0.2
	1000	0.6	0.2
	1250	0.1	0.2
	1500	0.01	0.2

Table 4-1: The final training scheme used for the clouds data set.

Concentric data

One would expect the concentric database (see Appendix B) to be an easy one to learn since the two classes it contains are completely separable. Unfortunately this is not the case: it appeared that a neural network trained for 1000 epochs with a constant learning rate of 0.6 on certain training sets hardly ever reached a performance above about 63%. And, what a coincidence, class 2 of the concentric database (the 'outer class') happens to contain about 63% of all the data. In other words, the network simply had learned that classifying all the given data as class 2, is an easy and fast means to gain a performance of more than 50%. But since it was only occasinally able to find anything like a class border that would lead to a higher performance, most of the time it felt back to the easy way: classifying all data to class 2. To circumvent this problem we tried some 'ridiculous', initial values for the learning rate and momentum and they seemed to solve the problem (more or less). As for the clouds data the number of hidden neurons did really seem to matter so we used the architecture as given in Figure 4.3.

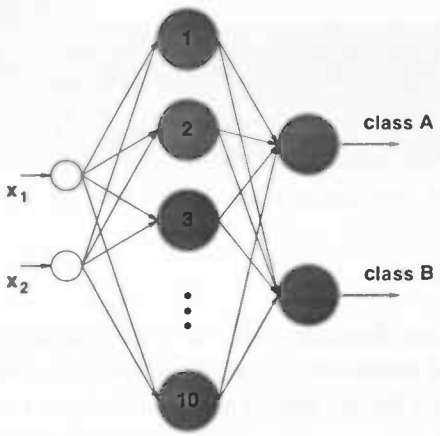


Figure 4.3: The network architecture used for the concentric database.

For the final scheme of training, to be used for the following experiments, see Table 4-2. For a typical learning curve when using this scheme see Figure 4.4.

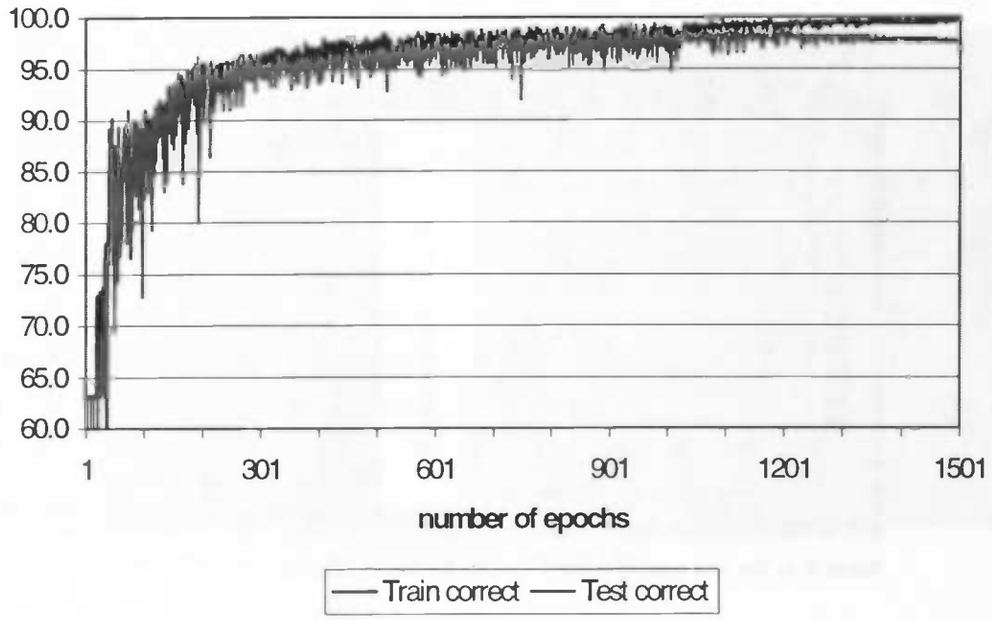


Figure 4.4: Typical learning curves for the concentric data set.

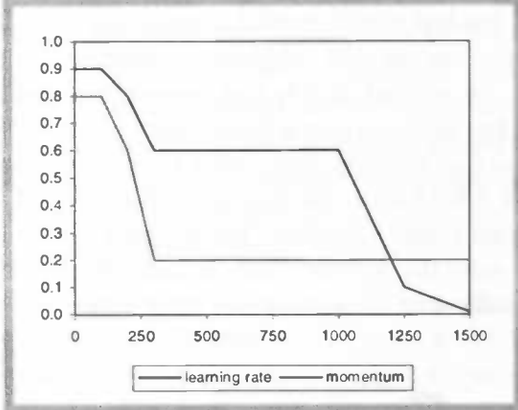
plot	data points		
	epoch:	learning rate:	momentum:
	0	0.9	0.8
	100	0.9	0.8
	200	0.8	0.6
	300	0.6	0.2
	1000	0.6	0.2
	1250	0.1	0.2
	1500	0.01	0.2

Table 4-2: The final training scheme used for the concentric data set.

Number plates data

The number plates database gave no trouble at all, as long as its size (it's got 30 inputs and 36 outputs) and therefore, its learning speed is not considered. So we have more or less arbitrarily chosen for 20 hidden neurons (we *are* talking about a lot of in- and outputs, so a little more hidden neurons cannot do much harm). See Figure 4.5 for a of the scheme used architecture.

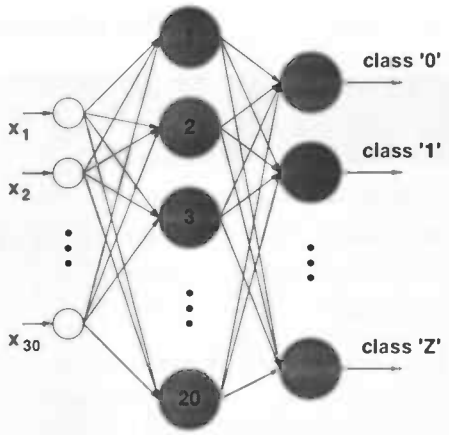


Figure 4.5: The network architecture used for the number plates database.

And since it is taking so much time per epoch we have decided to train for 'only' 800 epochs in total, which should still be more than enough since it learns quite quickly. See Figure 4.6 and Table 4-3 for a typical learning curve and the final training scheme.

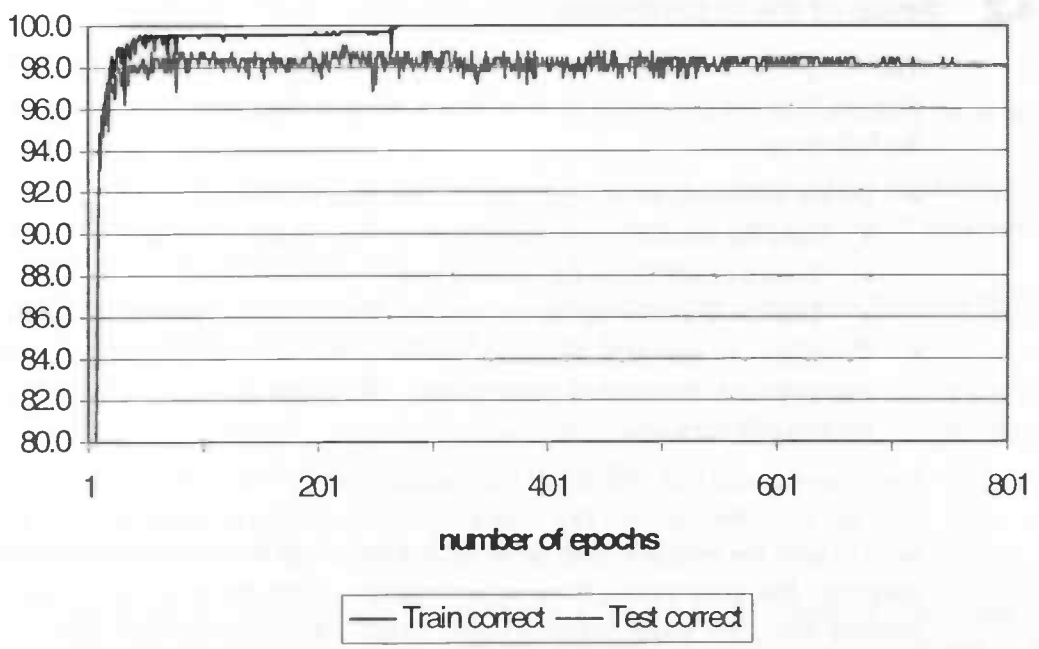


Figure 4.6: Typical learning curves for the number plates data set.

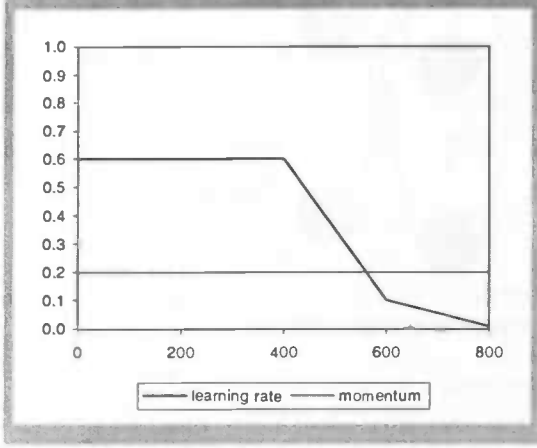
plot	data points		
	epoch:	learning rate:	momentum:
	0	0.6	0.2
	400	0.6	0.2
	600	0.1	0.2
	800	0.01	0.2

Table 4-3: The final training scheme used for the number plates data set.

4.2 Setup of the experiments

The test procedure to estimate the stability of a neural network classifier we will follow, is derived from the procedure by Hoekstra et al. (see paragraph 2.1). Our procedure will be of the following form:

- Do the following steps a number of times (say ten times):
  - Split the available set of data in a training set and a test set;
  - Train a classifier on the training set;
  - Test the classifier on the test set and determine its performance (or error rate);
- Calculate the standard deviation (and/or other statistic interesting features one can be interested in) of the set of performances. This, now gives an indication of the stability of the classifier at hand.

Only, how should this splitting of the available data set take place? Or, is it even necessary to split the available data into two separate sets? According to Hoekstra et al. (1997) there is no need to split the available data set at all; we could just as well use the training set for testing purposes. But even worse, there is, according to Hoekstra et al., no reason to use different training sets; only using different initial weights and a randomized order of presentation of the patterns each epoch should do the job! This would be very undesirable indeed. We would rather see that the stability of a classifier can (almost) fully be controlled by the size of data sets used for training (and for testing) instead of that badly chosen random initial settings can have a severe influence on the stability as well. It is, namely, simply not feasible to select an optimal (or even sub-optimal) random weight setting and random order of presentation of patterns at forehand. Our first series of experiments will therefore be aimed at getting some insight in what the real effects of these random settings are on the stability of a classifier. We hope to find some evidence for our assumption that the influence of these random settings on the stability is negligible, so that we may as well ignore them when concentrating on the effects of the size and composition of the data sets on the stability, which will be examined in the second series of experiments.

Experiments with the same training set

With the first series of experiments we will examine the effects of random initial weight settings and random order of presentations of patterns each epoch on the stability of neural network classifiers. We will do this by keeping the size and composition of the training and test sets constant for each experiment. We will, however do a series of experiments with varying sizes of the training and test sets to see if any effects that might show up have just as much impact on smaller training sets as on larger ones. Here is the scheme that will be followed:

- Split all available data in a working set of 1000 patterns and a large independent test set (while keeping the class ratios intact).
    - Do for each size of training set you are interested in:
        - Randomly split the working set in a training set (of the chosen size) and a corresponding validation set (again while keeping the given class ratios intact);
        - Do a number of times (say ten times):
            - Train a network on the training set;
            - Test the network on the following sets: training set, validation set and test set and determine the corresponding performances.
- Plot the results per size of training set used (in boxplots).

The size of the working set (1000 patterns) is chosen for a number of reasons:

- Since a lot of networks need to be trained the training set should preferably be as small as possible.
- It still allows us to split it in fairly large training (and corresponding validation) sets (compared to the average size of training sets that are used for most experiments described in the articles and books included in References)
- It also enables us to keep a fairly large, independent test set aside, that then can be used to give a relatively accurate estimate of the 'real' stability of a given classifier.

The networks are tested on various sets of data to enable us to examine whether any this makes any real difference. If it does make difference than the results of the (large,) independent test set will be considered the most accurate.

Experiments with a randomly chosen training set

The second series of experiments is meant to check the theory in Chapter 2 concerning the relation between the size of the training and test sets and the stability or our estimate of the stability of the trained classifiers. The procedure we will follow this is the following:

- Split all available data in a working set of 1000 patterns and a large independent test set (while keeping the class ratios intact).
    - Do for each size of training set you are interested in:
        - Do a (large) number of times (say twenty times):
            - Randomly split the working set in a training set (of the chosen size) and a corresponding validation set (again while keeping the given class ratios intact);
            - Train a network on the training set;
            - Test the network on the following sets: training set, validation set and test set and determine the corresponding performances.
- Plot the results per size of training set used (in boxplots).

The size of the working set is again chosen equal to 1000 patterns for the same reasons as for the previous series of experiments.

In the results of these experiments we hope to see some evidence about whether Fukunaga's theory (see 2.2) also holds for neural network classifiers. We will therefore look for the following patterns in the results:

- As the size of the training set becomes larger, the average attained performance should also become higher (or should remain constant, if the maximum attainable performance is already reached).
- As the size of the validation set becomes smaller, the variance over the attained performances should become larger.
- The average attained performance determined when using the validation set for testing should approximately equal the average performance determined when using the test set for testing.
- The variance over the attained performances, when using the test set for testing should remain approximately constant for the various sizes of the training set.

4.3 Results of the experiments with the same training set

Clouds data

For plots showing the results of the experiments with the clouds database see Figure 4.7. Per bar 10 classifiers are trained. In these plots on the horizontal axis, the size of the training set is given in a percentage from the working set. We can see a few things in these plots:

- The first thing that becomes apparent, when looking at the plots, is that it the random initial weight settings and the random order of presentation of patterns do seem to have an effect on the resulting performance of the classifiers.
- It also does seem to matter, which data set is used for testing. Therefore we will assume that the lowermost plots, when using the test set for testing, give the most accurate picture. But even when looking at these lowermost plots we must accept that a maximum difference in performance of about 1 percent is inevitable.
- Since we have only used one composition per size of the training set, we cannot really draw any conclusions like that the size of the training set is irrelevant to the stability.

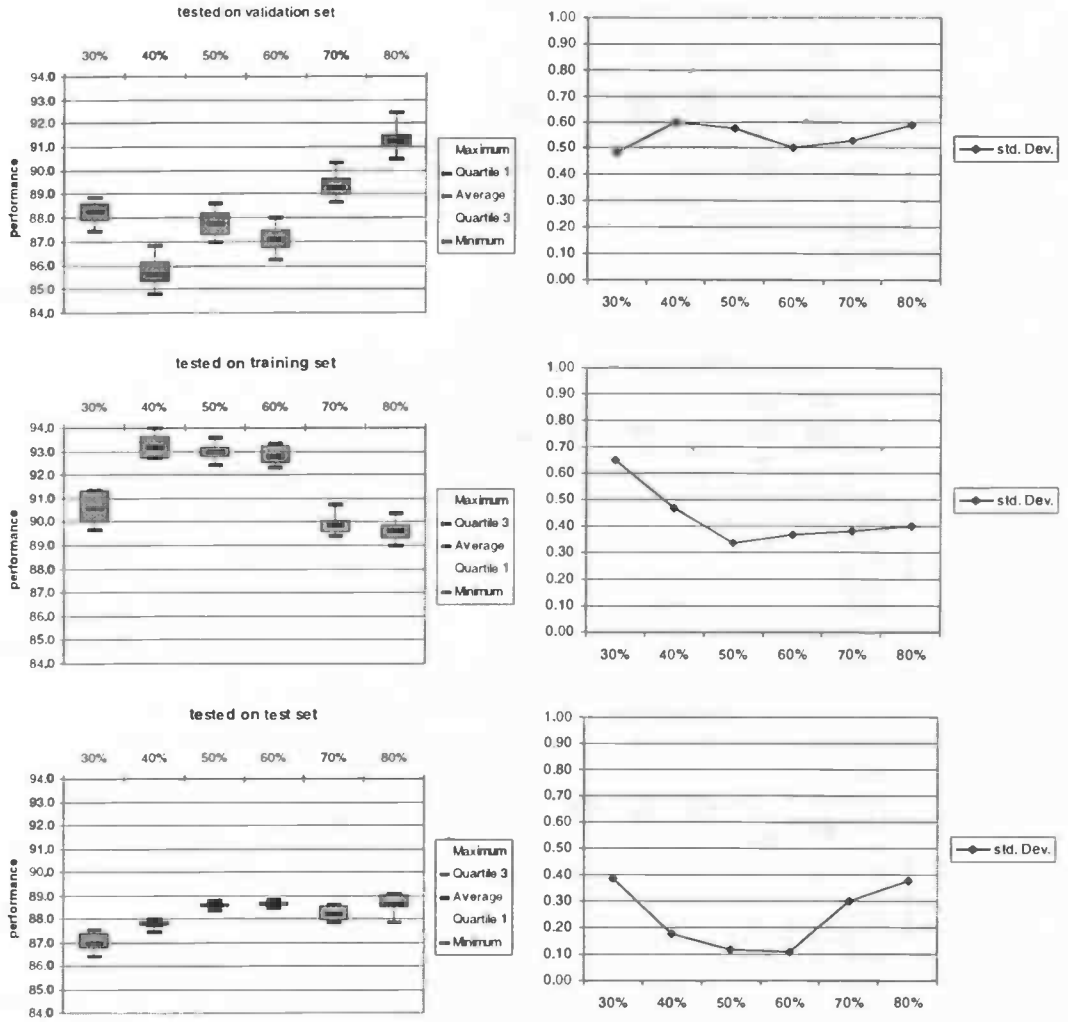


Figure 4.7: Results from experiments with the clouds database using the same training set. Note that the effect of randomly chosen settings is not negligible and that testing the same classifier on different test sets give different results.

Concentric data

See Figure 4.8 for the results. This time we have trained 15 classifiers per bar (since it seemed at first like we had a somewhat higher standard deviation compared to the experiments with the couds data). For the rest a somewhat similar story holds as for the experiments with the clouds data:

- Again, it becomes apparent, when looking at the plots, that the random settings do seem to have an effect on the resulting performance of the classifiers.
- It also does seem to matter, which data set is used for testing. But, this time, the lowermost plots even show a maximum difference in performance of about 3 percent (and a average of about 2.5 percent).
- And again, it *seems*, like the size of the training set is irrelevant to the stability.

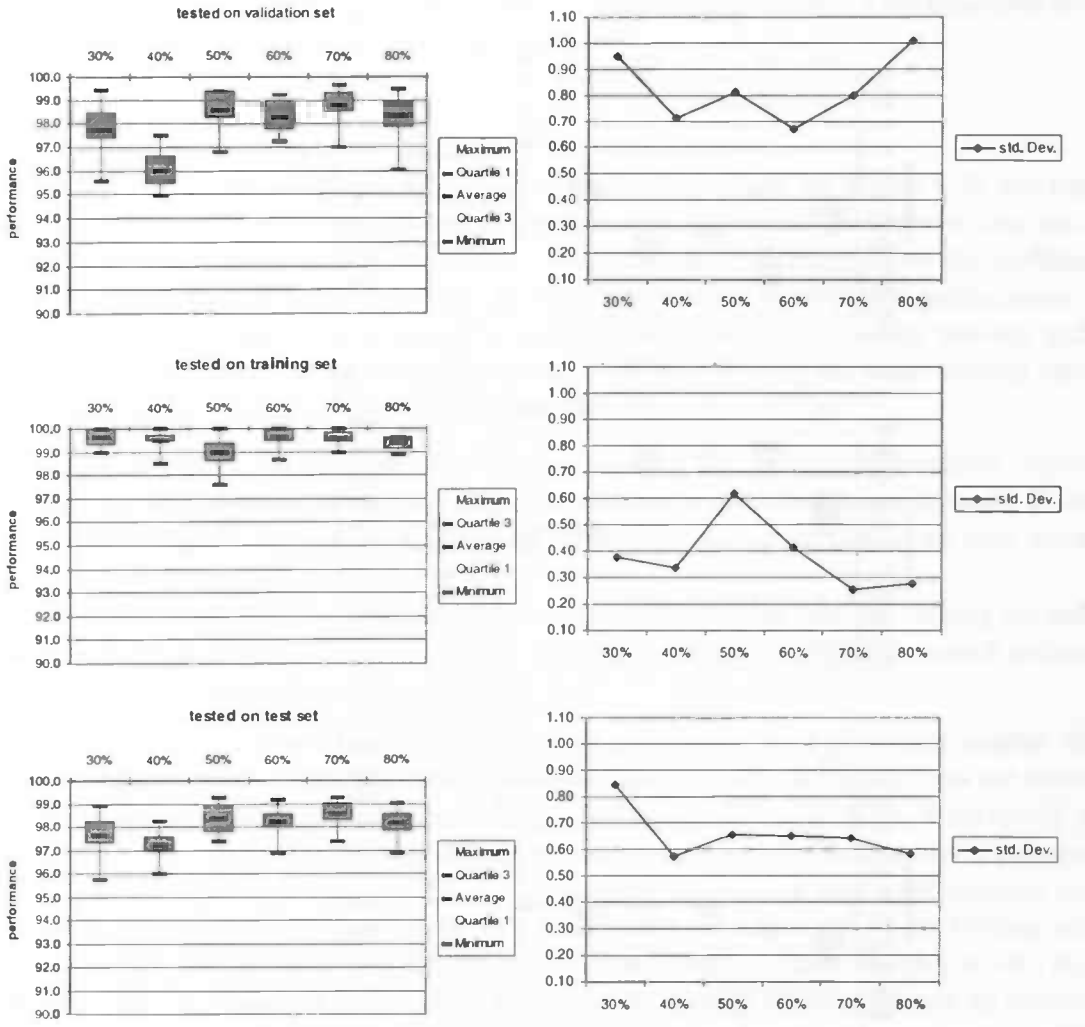


Figure 4.8: Results from experiments with the concentric database using the same training set. Even though both classes ought to be completely separable a performance of 100% on the training set is only sparsely obtained. And even though we have trained an extra 5 classifiers per bar (compared to the clouds data) both the maximum and the minimum variance on the test set are far from being negligible.

Number plates data

The plots of the results of the experiments with the number plates data are given in Figure 4.9. Per bar we have trained 10 classifiers. Only this time it seems like the influence of the random settings is truly negligible (when looking at the lowermost plots, where the test set is used for testing).

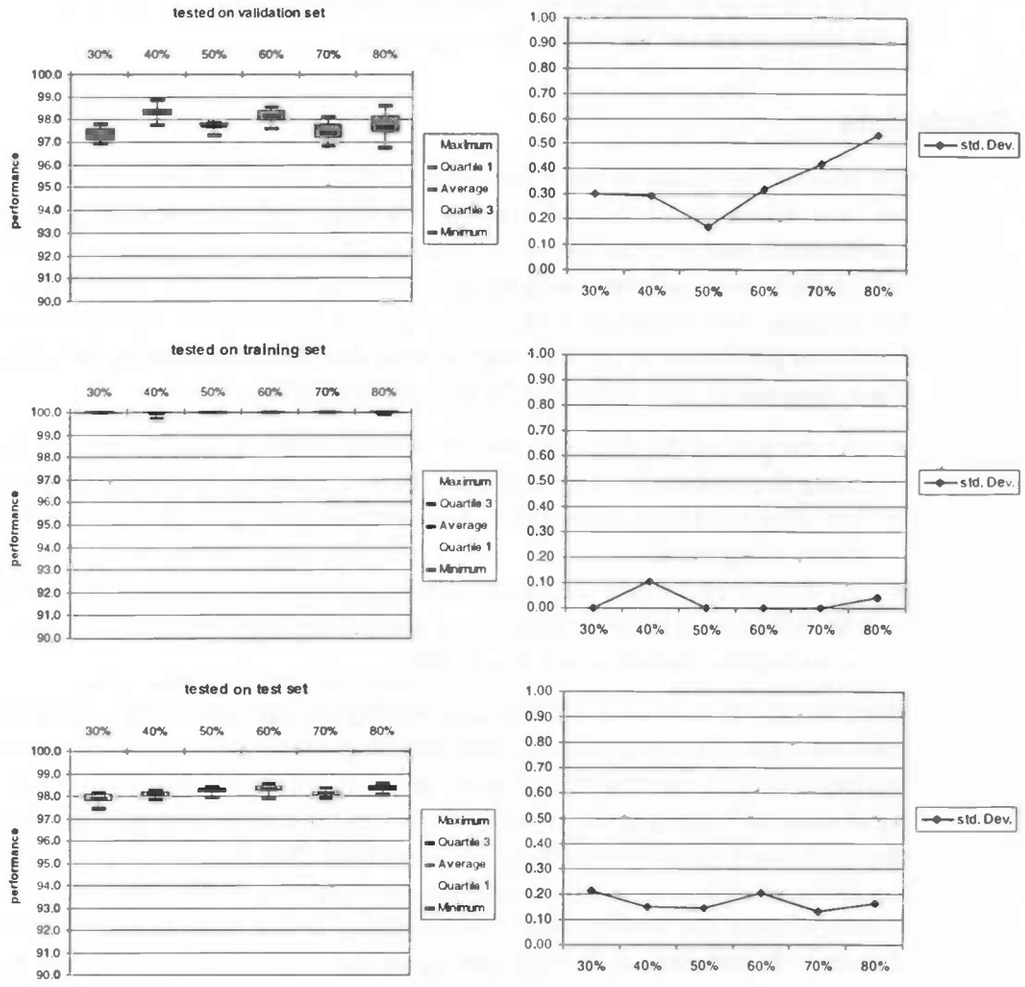


Figure 4.9: Results from experiments with the number plates database using the same training set. This time with only very few exceptions (two or three at most), the maximum performance is ottained on the tralning set. As a result the tests the other test sets show aloso only a minimum amount of variation (with the exception of the smaller validation sets. They simply appear to be too small to give any sensible results).

All in all, one conclusion that can be drawn from the above results is that the effects of random initializing the networks starting weights and the randomized order in which each epoch the patterns are presented, are not entirely negligible. But, as stated before, since it is not feasible to determine, at forehand, an optimal (or even sub-optimal) set of starting weights or a preferred order in which to present the various training samples, we simply have to learn to live with it. Of course it may be that most of this problem dissolves itself when training sets are used that are significantly larger than the ones used in the above-described experiments, but that would require a significantly larger set of data to be available (and it would, of course, be much more time consuming).

4.4 Results of the experiments with a randomly chosen training set

We are, amongst others, interested in whether it is possible to draw conclusions from tests using only a limited amount of data (in our case: the working set) that can normally speaking only be drawn when a significantly larger set of data is available. Therefore, for each database, the results of the experiments with randomly chosen training sets will be presented in two seperate figures. First we will take a look at the plots for which only the working set

of data was used and then we will compare these results with the results of the tests with the large independent test set that we have kept aside.

Clouds data

For plots of the results of the experiments with the clouds database see Figure 4.10. Per bar we have trained on 20 different training sets which were then tested on different data sets. On the horizontal axis the size of the training set is given in a percentage from the working set. In the lowermost plots, with the title 'training performance – validation performance', the statistics are calculated over the differences between the corresponding training and validation performances on the same network that was trained using the same training set.. Now, what can be seen on/concluded from these plots?

- As the size of the data set used for training increases, the average performance, when using the validation set for testing, does indeed show a somewhat up-going trend. If on the other hand the training set is also used for testing the opposite is true: we now see a down-going trend.
- As the size of the data set used for testing decreases (counts for both the training set and the validation set), the variance over the attained performances increases, which makes as a result the classifier look less stable.

Even though we have done a fairly large number of experiments for a quite large number of different sizes of training sets (and corresponding validation sets), the plots still do not show the monotonously increasing/decreasing trends as we would like to see them; there is still a lot of somewhat unexpected variance. A questions that might arise is whether this is perhaps due to (a minor form of) overfitting. We would therefore like to know how a performance on a training set relates to a corresponding performance on a validation set. As the training set becomes larger, one would expect that overfitting is less likely to occur. In other words, the difference in performance on each training set and on the corresponding validation set should show a smaller variance. Instead, if there is any patterns to be found in these plots, it looks most like the variance is increasing when larger training sets are used. Besides would't overfitting go along with some more extremely high performances on the training set?

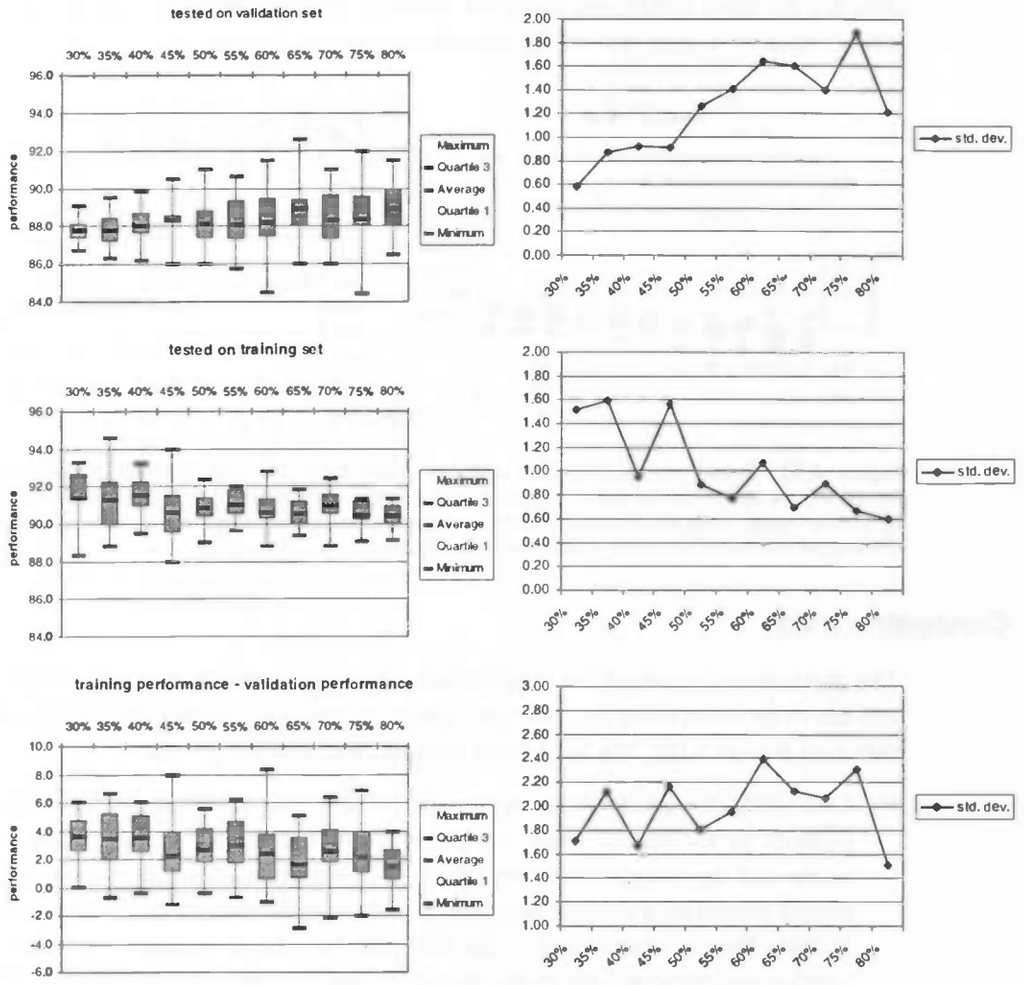


Figure 4.10: Results from experiments with the clouds database using randomly chosen training sets. In the upper left plot we can see a slightly up-going trend in average performance as the training set becomes larger. Somewhat more clearer to see is that as the set used for testing becomes smaller the variance becomes larger, which makes the classifier look less stable. It is also clear from these plots that we have not encountered any (serious) overfitting problems.

Next we are going to compare the results on the working set alone with the results of testing the same classifiers on the large, independent test set. The results of the tests on this test set are given in Figure 4.11.

- First of all, it seems like the average estimates of the stability when using the validation set for testing does indeed come close to the average estimates of the stability when we are testing the classifiers with the large test set.
- Second, the variance over the estimates of the performances when testing with the test set looks pretty stable itself. It is significantly smaller than the corresponding variances when using the training and/or validation set for testing. It even comes close to the maximum variance that was found when using the same training set for experiments (see Figure 4.7). And the only trend it shows is a slight decrease as the size of the training set increases. Apparantly the size of the training set has also a slight influence on the stability of the resulting classifier.

Note that due to the use of a limited test set (like the validation or the training set), one might be inclined to, erroneously, come to the conclusion that the classifiers trained on the clouds

data are far from stable and are quite sensitive to the composition of the training set while when tested on a large test set the opposite appears to be true.

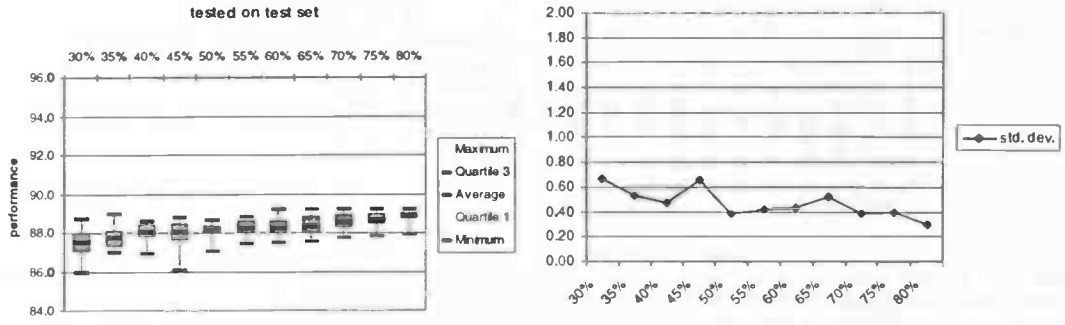


Figure 4.11: Results from experiments with the clouds database using randomly chosen training sets. When comparing the above results with the tests on the validation sets in Figure 4.10 the average performance seem to be closely related. Only the standard deviation appears to be significant smaller. We also see that as the size of the training set increases, the stability also slightly increases.

Concentric data

The plots of the results of the experiments with the concentric data are shown in Figure 4.12 and are to be interpreted in a similar fashion as the plots of the experiments with the clouds data (see Figure 4.10). We have again trained 20 classifiers per bar.

- Like with the clouds data, as the size of the data set used for training increases, the average performance, when using the validation set for testing, does indeed show a somewhat up-going trend. But, if on the other hand the training set is also used for testing purposes, a straight line appears (close to the maximum attainable performance of 100%). This is mostly due to the fact that the clouds database contains no regions of overlap and therefore, on each training set the maximum performance of 100% should be attainable.
- As the size of the data set used for testing decreases (counts for both the training set and the validation set), the variance over the attained performances remains about constant, which might lead to the conclusion that even the smallest test set is large enough to make a reliable estimate of the stability of a classifier trained with the concentric database.

And, finally, the lowermost plots show that, as the training set increases, the difference in performance between the tests on the validation set and tests on the training set becomes smaller (how surprising) but the variance on the other hand remains constant.

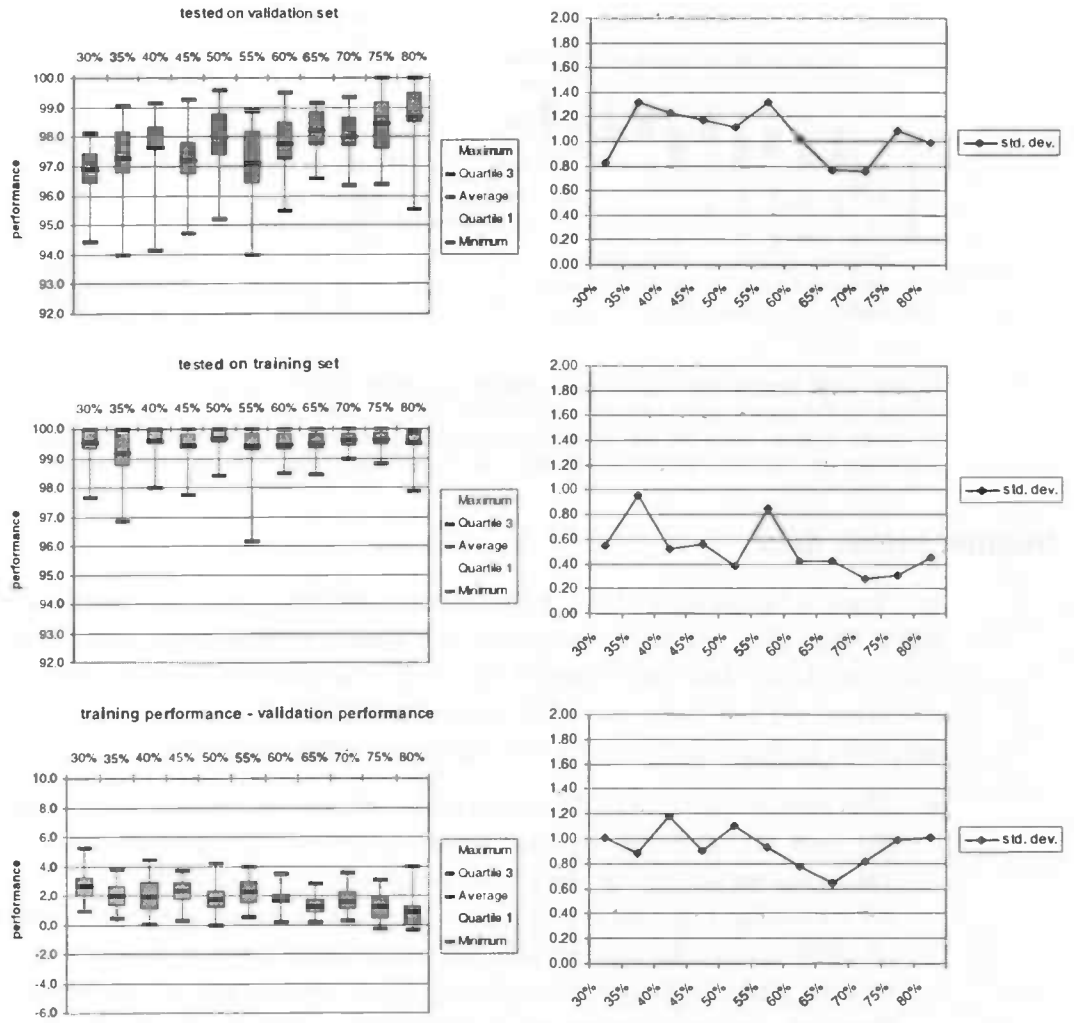


Figure 4.12: Results from experiments with the concentric database using randomly chosen training sets. The only trend that becomes visible is the somewhat increasing average performance as the training set becomes larger. For the rest it are only straight lines, so no signs of overfitting either, for instance on the smaller training sets.

Now we will compare the results of the experiments on the working set alone with the results of testing the same classifiers on the large, independent test set. The results of the tests on this test set are given in Figure 4.13.

- Again (as with the clouds database) it seems like the average estimates of the stability when using the validation set for testing does indeed come close to the average estimates of the stability when we are testing the classifiers with the large test set.
- The variance over the estimates of the performances when testing with the test set does not look that stable, though. It actually looks pretty much like the results of the tests when using the corresponding validation set for testing. This agrees with our conclusion that it seems like the smallest set we have used for testing (a validation set of 200 patterns) is large enough to give a reasonable estimate of the stability of a classifier trained on the concentric database. And finally it appears (as with the clouds data) that the variance of the resulting performances of when using the test set for testing comes close to the maximum variance that was found when using the same training set for experiments (see Figure 4.8).

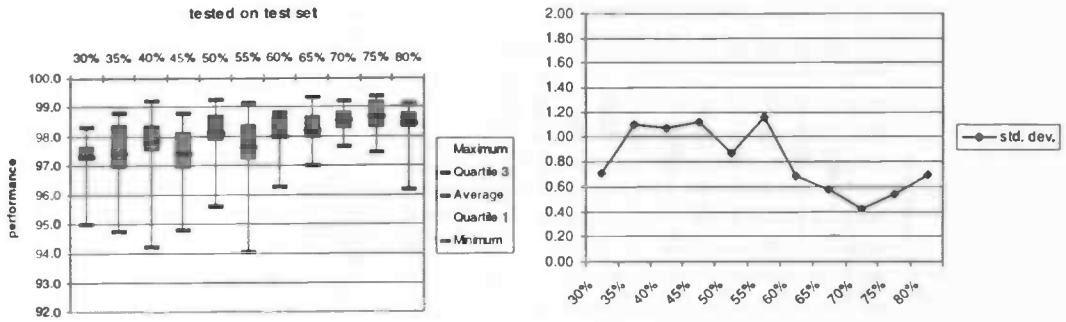


Figure 4.13: Results from experiments with the concentric database using randomly chosen training sets. When comparing the above results with the tests on the validation sets in Figure 4.12 the average performance seem to be closely related. Even the standard deviation appears to be not much approved by using this larger test set. Apparantly any instability problems with this database are mostly due to the training process.

Number plates data

In Figure 4.14 the plots of the results of the experiments with the number plates are given. Again these plots are to be interpreted in a similar fashion as the plots of the experiments with the clouds data (see Figure 4.10). Only this time we have trained no more than 15 classifiers per bar (also note that there are less bars) since not much variation in the outcomes appeared and since training classfiers on this database is very time consuming.

- This time, as opposed to the results on the clouds and the concentric data, as the size of the data set used for training increases, the average performance, when using the validation set for testing, does again not show any trend at all. When using the training set for testing a straight line appears even close to the maximum attainable performance of 100%, as was the case with the concentric data. This is simply due to the fact that the number plates database contains no regions of overlap and is relatively easy to learn and therefore, on each training set a performance close to the maximum performance of 100% is reached.
- As the size of the validation set decreases, the variance over the performances increases. However, nothing like this occurs when testing the classifiers on smaller training sets. when testing on the training set there is simply no resulting variance. This is for the same reason as why the average performance shows a straight line at approximately 100%.

And, finally, the lowermost plots is just an exact copy of the right most plot (variance on the tests with the validation set) and has therefore nothing new to tell.

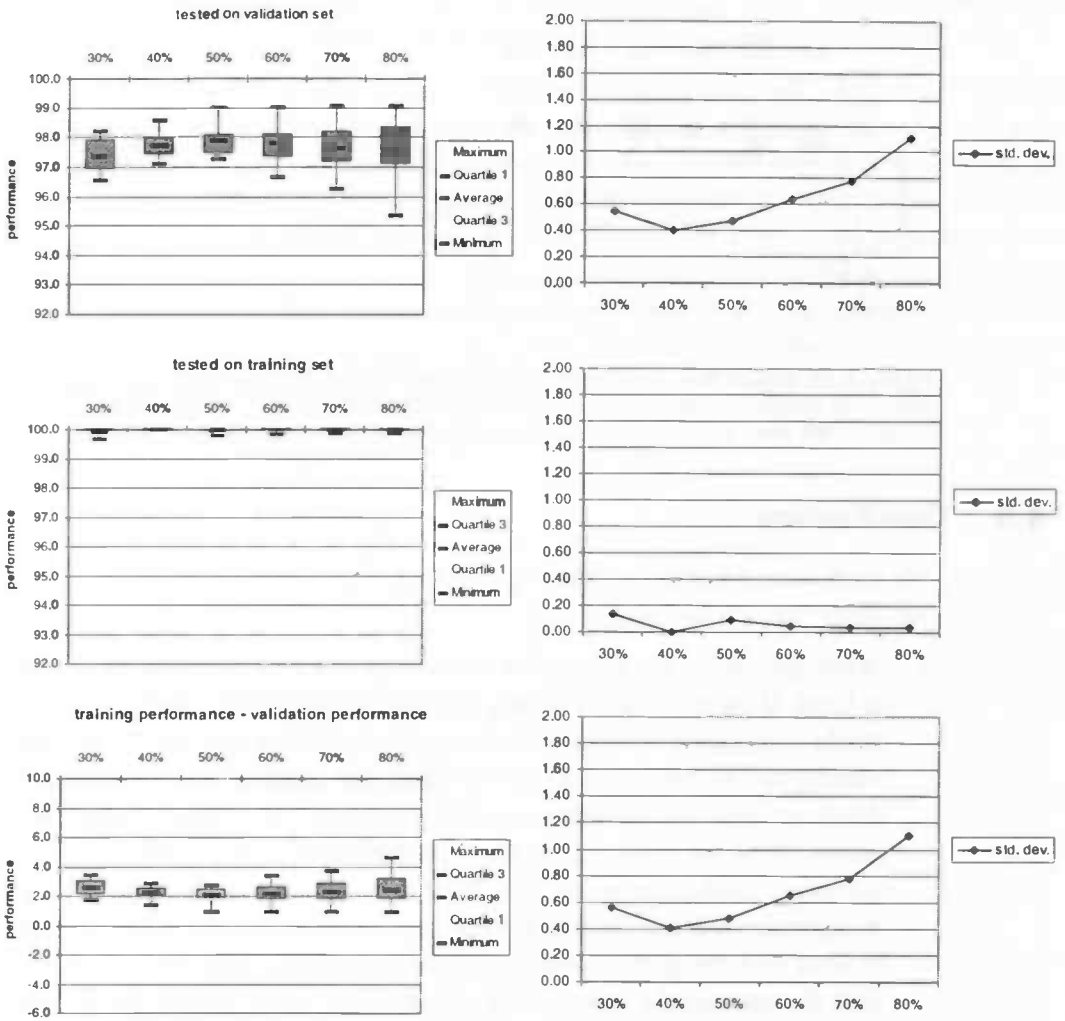


Figure 4.14: Results from experiments with the number plates database using randomly chosen training sets. See how on practically each training set the maximum performance was attained. The only variation that shows is as the validation set becomes too small.

And once again we will compare the results of the experiments using only the working set with the results of testing the same classifiers on the independent test set that we have kept aside. The results of the tests on this test set are given in Figure 4.15. Here is what can be concluded:

- In contrast to the results of the experiments with the clouds and concentric data, the average estimates of the stability when using the validation set for testing does not correspond to the average estimates of the stability when testing the classifiers with the larger test set. Well, at least not as the validation set becomes smaller in size. A possible explanation might be that since this database contains a large number of classes with different numbers of patterns belonging to each of them, some 'easy' classes might not be represented anymore in the validation set, while the training sets are still not large enough to fully learn the more 'difficult' classes.
- The variance over the estimates of the performances when testing with the test set looks pretty stable. It only shows a slightly decreasing trend as the size of the training set increases. It even seems like the variance becomes smaller than the maximum variance that showed for the results of our experiments on the same training set (see Figure 4.9).

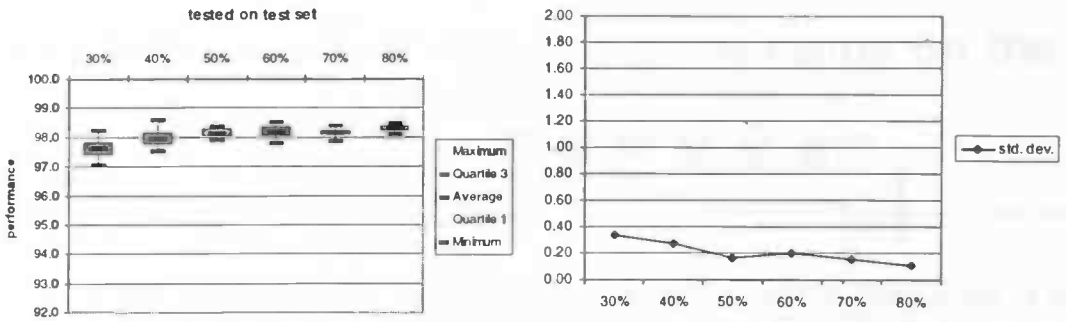


Figure 4.15: Results from experiments with the number plates database using randomly chosen training sets. This time when comparing these plots with the tests on the validation sets in Figure 4.14 it appears that the smaller validation sets were simply to small for even estimating an accurate average performance.

4.5 Conclusions

We have done a number of experiments with three quite distinct data sets and have come to the following conclusions:

- When estimating the stability of a classifier it is best to have a large independent test set at hand. If the training set is easy to learn (number plates data) then the results looks too stable (compared to tests with a large independent test set). If on the other hand the training set is somewhat harder to learn (for instance due to overlap as in the clouds database) then the resulting standard deviation on relatively small training sets is far greater than the resulting standard deviation on a large test set. And, of course, the (average) performance on the training sets is usually much higher than on an independent test set. Only as the training sets that we have used, became larger it started to look like the performances and stability on the train sets showed more similarities with the tests on the independent test set. This would not be that illogical, but since we have not experimented with any larger sets of training data, it is pure speculation.
- The effects of the random choice of initial weights and the random order in which the patterns are presented to the neural network classifier each epoch can have a more than negligible influence on the stability of the classifier. Depending on the 'hardness' of the problem at hand, this might result in maximum differences in performance up to 2.0 %. But unfortunately it is simply not feasible to select proper setting at forehand.
- According to our experiments, when only a limited amount of data is available, it is simply impossible to tell whether a variance in the estimates of its performance is due to the limited size of the test set or due to the limited size of the training set (or both). In other words, in such cases, it is practically impossible to get an accurate estimate of the 'real' stability of the classifier at hand.
- As we have seen in the above results, it is possible to get a relatively accurate estimate of the 'real' average performance of a classifier at hand with only a limited amount of data available. Therefore a number (preferably a large number) of experiments must be carried out with random chosen trains and test sets (for each experiment). The so obtained average performance will most likely be a very reasonable estimate. But unfortunately it is hard to say how many experiments are needed. And it does not tell us how much of the spread in results (that is how much of it's apparent instability) is due to the limited size of the test sets. Note that this procedure has a lot in common with well-known procedures as *bootstrapping* or *k-fold cross-validation*. Especially k-fold cross-validation seems like a neat way to get (nearly) any number of independent training and

tvalidation sets (For more information on these two methods see, for instance the following ftp-site maintained by W.S. Sarle: "ftp://ftp.sas.com/pub/neural/FAQ.html").

All in all it is surely possible, by doing the right experiments the right number of times (that is, *many* times), to gain some usefull information about the 'real' stability and performance than that are attainable if a lot more data is available. But that not only requires a lot of time consuming experiments (note that we have done quite a number of experiments) it may also not be just as reliable for every set of data. Note that we have only done experiments with three distinct databases and have seen different results for each of them. So it is always possible that for certain other clasification problems things will be a lot easier. And if the classifier at hand appears to be stable enough on small test sets, then it can only be even more stable on larger test sets.

Chapter 5 Influence of varying class ratios on the stability

So far we have concentrated on the effects of the size and accidental composition of the training (and test) set on the stability of the resulting classifier. However, so far we have always kept the given class ratios intact. This is the customary use, to treat the proportions in the given data set as a priory chances and therefore keep them constant, even while the theory may demand the use of truly random chosen data sets (see, for instance, Duda and Hart, 1973). In this chapter, however, we are interested in the influence of varying class ratios on the performance of the resulting classifier. It can be important to know about these influences when working with a data set that is not guaranteed to contain a class ratio that directly corresponds to the a posteriori class probabilities. If in this case the resulting classifier happens to be extremely sensitive to any change from these given class ratios, it will be hard (if not impossible) to obtain a reliable estimate of its real performance.

This time we will also take a look at classifiers with two hidden layers and see if there is any difference in stability between such a network architecture and the single-hidden layer networks we have been using so far. Well, actually we have already done some experiments with two hidden layers, but they did not seem to be able to outperform a network with a single hidden layer at any of the databases we have been using. They only seemed to learn slower: on average they need more training epochs to obtain the same performance. Still, networks with two (or more) hidden layers are used a lot, even for (simple) problems like ours. And almost without an exception, no explanation is given for the choice of the architecture (just like we did not fully explain the architectural choices in Chapter 4).

In this chapter we will first introduce a possible test procedure to estimate a given classifier's sensitivity to varying class ratios. We will then compare the results of our test procedure with the 'raw' classifier outputs to see how the output of our test procedure correlates with the exact position of the class boundaries as our classifier has learned them. Then we will do a number of experiemnts with some classification problems without overlap. We expect (and hope) to see that on such classification problems it does not really matter whether the class ratios in the training and test set exactly match or not. And finally we will take a somewhat closer look at the effects of the size of the training set on the class sensitivity. This will surely have some influence, but can such results still be used if we are not concerned with test results that are accurate within a range of one percent?

5.1 Test procedure

What we need is a test procedure that is preferably easy to perform and whose results are even for multi-dimensional and/or multi-class problems easy to interpret. This test procedure should not rely any class ratio as a reference, for instance on the ratio in the data set at hand, since we are sure in what way this ratio corresponds to the 'real' (a posteriori) class probabilities. Therefore an overall performance measure as we used in Chapter 4 is not an appropriate test procedure. Statistics as the Sensitivity, Positive Predictivity and False

Positive Rate can provide more meaningful results (see Lawrence et al., 1998). These measures are defined on a class by class basis as follows:

- The **Sensitivity** of a class is the proportion of patterns labeled as belonging to that class which are correctly classified by the network. That is equal to the global performance of the classifier when the test set consists only of pattern belonging to the class at hand.
- The **Positive Predictivity** of a class is the proportion of patterns which was correctly predicted to belong that class.
- The **False Positive Rate** of a class is the proportion of patterns that actually belong to other classes but that were incorrectly classified as belonging to the class at hand.

The sensitivity criterium is the one that seems to be best suited for our problem at hand. It is even possible to use this criterium to create a single performance measure for the whole classifier (instead of having a measure per class). This is what Lawrence et al. (1998) do as they introduce a mean squared sensitivity error (MSSE) which is defined as follows:

$$\mathrm{MSSE} = \frac{1}{N_C}\sum_{i=1}^{N_C}(1 - S_i)^2$$

where N_C is the number of classes and S_i is the sensitivity of class i as defined above. Even though there may be are other ways to define a single performance measure, they have among others things, as a drawback that they tend to hide possibly vital information about which single classes are actually responsible for a higher error rate. We will therefore instead, use the sensitivity measure as in the following procedure:

- Do for each class (that you are interested in):
  - Do for each percentage you are interested in:
    - Train a network with a training set of a fixed size, containing the given percentage of patterns belonging to the class at hand while keeping the other class ratios constant at the given class ratios (with respect to each other).
    - Test the network on patterns belonging to the class at hand only and determine the performance (or error) rate by aplying the winner rule (highest output wins, no rejections).
  - Plot the results in a single plot.

The resulting plot should then be interpreted as follows: if on two (or more) following input points there is no difference in output, then the classfier at hand is unsensitive to the variations in class ratios within that input range. But, since even random settings of classifiers can cause the resulting overall performance to fluctuate within a range of a few percent (see Chapter 4) one can not expect to find exactly equal results on any two following input points.

It is hard to say at forehand what kind of plot we can expect to see when applying this procedure on an arbitrary classification problems. But there is one common thing that we expect to see every plot, namely we expect all plots to be monotonous rising. Since if we are training a classifier with more data belonging to one class then we would surely like to see that this classifier becomes better at classifying unseen patterns belonging to this same class.

According to this procedure we will do all training with training sets of the same fixed size. This will probably mean that if we have a given data set, once we have chosen a size for the training sets we might not have enough patterns belonging to one or more classes when

changing the ratios. We will circumvent this problem by simply allowing the same patterns to occur more than one time in the training set. Now this can be done on several ways, for instance by bootstrapping the original training set, that is, use (per class) random sampling with replacement. But this might lead to some patterns to occur N times in the training set while other do not occur at all. This can't be too good, especially with not with a small initial number of patterns belonging to the class at hand. We will utilize as little random sampling as possible. We will therefore first duplicate the entire set of patterns belonging to the class at hand as many times as possible and then use random sampling for the remaining number of patterns that is required to complete the total number of patterns we need. Though it not strictly necessary, we will also use this procedure for the following experiment with artificially generated data sets.

5.2 Results from test procedure vs. classifier outputs

We will now do a number of experiments on a few simple artificially generated classification problems To make a comparison between the results of our test procedure and the actual classfier outputs. Since the classifier outputs can only be visualized for problems with a low dimensional input space we will only look at one- and a two-dimensional problems.

One-dimensional classification problem with overlap

For the first experiment, a one-dimensional, two-class classification problem with overlap was used. The patterns belonging to both classes (class A and class B) are uniformly distributed according to the following conditional probability functions:

Class A:
$$f(\mathbf{x} \mid A) = \begin{cases} 1 & \text{if } 0.0 \le \mathbf{x} \le 0.6 \\ 0 & \text{otherwise} \end{cases}$$

Class B:
$$f(\mathbf{x} \mid B) = \begin{cases} 1 & \text{if } 0.4 \le \mathbf{x} \le 1.0 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{x}$ is the input vector.

A number of networks was then trained with 1000 training samples each at various class ratios. We expect (and hope) to see that the resulting classifiers always classify the non-overlapping part correctly (0.0 <= x <= 0.4 and 0.6 <= x <= 1.0). Only in the input region where the overlap occurs, it will (most likely) be inevitable that the classifier will make more errors on patterns coming from the class of which it has seen the least examples.

In Figure 5.2 the outputs of the ratio test procedure are shown. Some typical, corresponding output plots can be seen in Figure 5.3. In these latter plots the corresponding values at the two outputs are show for when the corresponding value at the horizontal axis is used for input. For both figures, the plots on the left side show the results when using a classifier with a single hidden layer, containing 10 neurons. For the plots on the left side a classifier with two hidden layers, containing 5 and 4 neurons respectively, was used. See for a graphical respresentation of the used network archectures Figure 5.1. The training set and the test set (for the sensitivity tests) both contained 1000 patterns. For the sensitivity test procedure we have thus used 500 patterns per class to testing. The default learning scheme used for these tests (and for some to come) is given in Table 5-1. This scheme might look a bit overdone

for our simple problems at hand (1000 training epochs is sure more than a sufficient number) but, perhaps unless overfitting is involved, it never hurts to do a few extra training cycles.

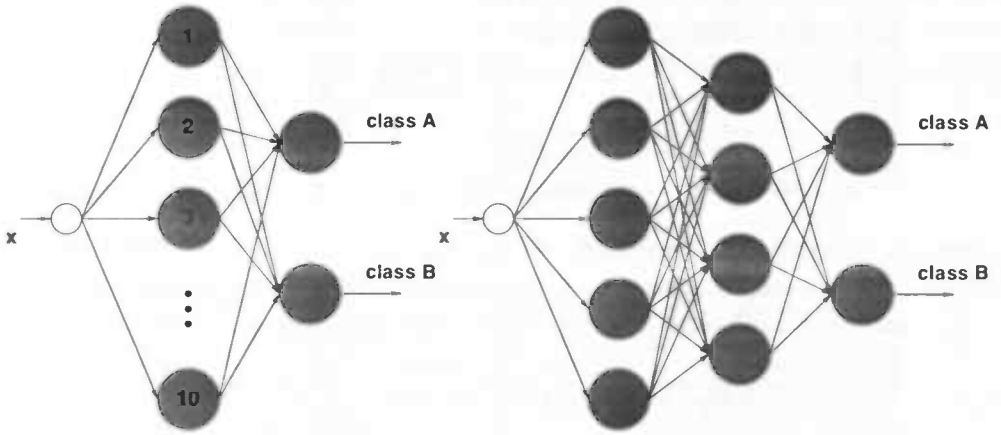


Figure 5.1: The network architectures used for the experiments on the one-dimensional classification problem.

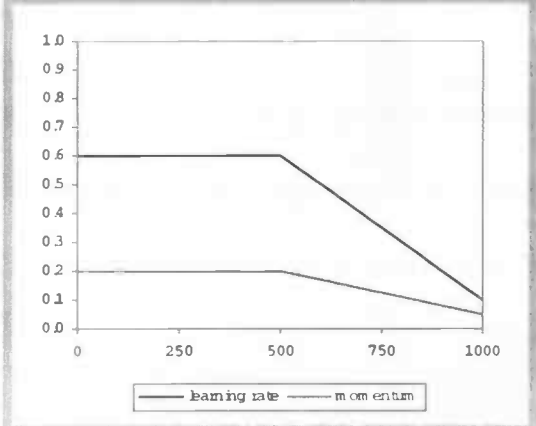
plot	data points		
	epoch:	learning rate:	momentum:
	0	0.6	0.2
	500	0.6	0.2
	1000	0.1	0.05

Table 5-1: The default training scheme used for the sensitivity tests.

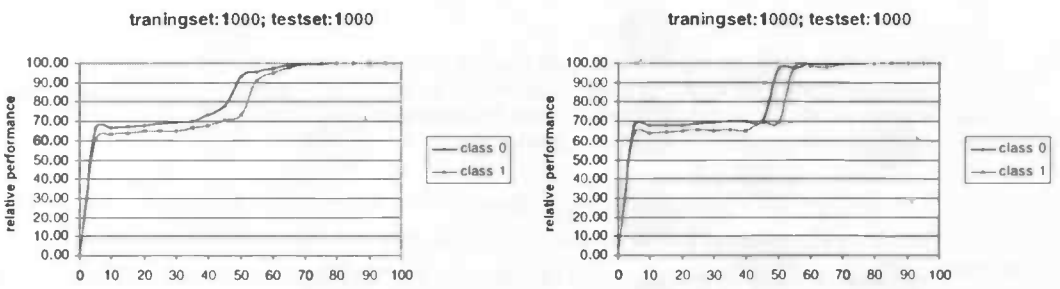


Figure 5.2: Results of the sensitivity test on the one-dimensional, two-class classification problem. For this simple classification problem it does not really make a difference whether we used a single (left) or two (right) hidden layers.

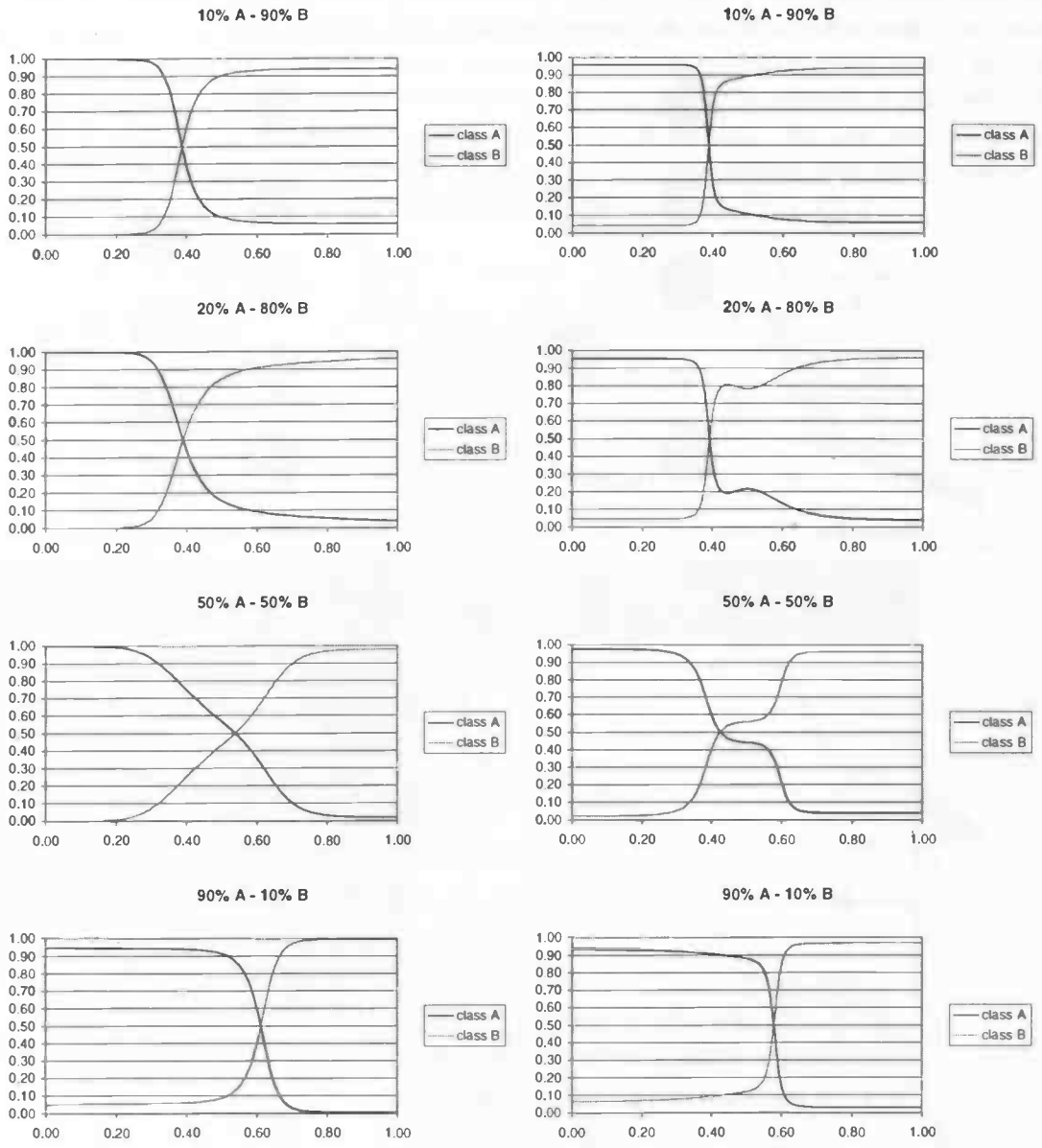


Figure 5.3: Each plot shows the outputs (vertical axis) of a classifier trained on classes A and B in the given ratio, corresponding to the input (horizontal axis). Note that the classifier with a single hidden layer (left plots) gives more smooth results, while the classifier with two hidden layers (right plots) seems to be better capable of learning the actual class probabilities at each input point.

Let us now take a closer look at these plots. In Figure 5.3, when simply taking the higher score as the winner (no rejections) it is clear to see that the classification result in the regions without overlap is pretty much independent of the used class ratio. When looking at the plots on the left side we also see a more grudual shift in the classification result as the class ratio in the training set changes than in the plots on the right side. These differences can also more or less be seen in the Figure 5.2, showing the results of the sensitivity tests. The horizontal parts in these plots thus seem to denote regions of little sensitivity to variations in the class ratios.

When looking closer at the differences between the results using the classifier with a single hidden layer and the classifier with two hidden layers the plots in Figure 5.3 show some interesting details. For instance: in the right plot obtained by using as much pattern from

class A as from class B (both 50%) all patterns in the 'overlap-area' are classified as from class B. This seems to be due to the fact that a classifier with two hidden layers is better capable of learning the actual class ratio at each input value (which is apparantly the case). If now, even though the overall class ratio is fifty-fifty, the actual class ratio within the 'overlap-region' is in favor of class B then, as a result, the classifier will learn this different ratio at these particular input values.

Two-dimensional classification problem with overlap

The second experiment was carried out on the following two-dimensional, two-class classification problem. The patterns belonging to both classes (class A and class B) are uniformly distributed according to the following conditional probability functions:

Class A:
$$f(\mathbf{x} \mid A) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ in circle}(0.35,0.3 \\ 0 & \text{otherwise} \end{cases}$$

Class B:
$$f(\mathbf{x} \mid B) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ in circle}(0.65,0.6 \\ 0 & \text{otherwise} \end{cases}$$

where $\text{circle}(x_1, x_2, r)$ is a circle at x_1, x_2 with radius r and where $\mathbf{x}$ is the input vector.

As a result the two classes overlap each other over a small area. A number of networks was then trained with 1000 training samples each at various class ratios. Again we expect to see that the resulting classifiers always classify the non-overlapping part correctly.

In Figure 5.6 the outputs of the ratio test procedure are shown. In Figure 5.5 some typical, corresponding output plots are shown. These latter plots should be interpreted as follows: each line corresponds to a specified difference in output values of the two output neurons corresponding to the input values, shown at both axes. The lines can be seen as contour lines and are drawn for the differences: -0.8, -0.4, 0.0, 0.4 and 0.8.

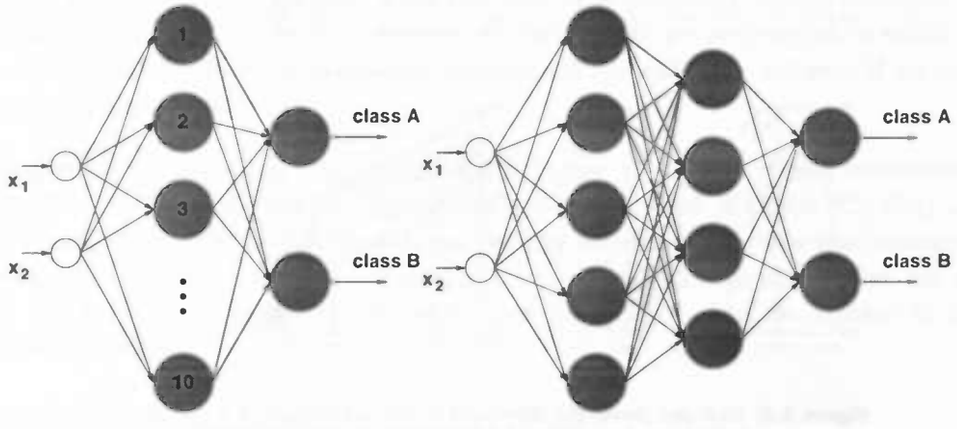


Figure 5.4: The network architectures used for the experiments on the two-dimensional classification problems.

The plots on the odd rows (the first and third row) show the results of training with a classifier with a only single hidden layer of 10 neurons (see the left plot in Figure 5.4). The plots on the even rows (the second and fourth row) show the results obtained by using a

classifier with two hidden layers, containing 5 and 4 neurons respectively (see the right plot in Figure 5.4). For the sensitivity plots: the plots on the left side show the results when using a classifier with a single hidden layer, containing 10 neurons and the plots on the left side when using a classifier with two hidden layers, containing 5 and 4 neurons respectively. The training sets for each experiment consisted of 1000 patterns. For the sensitivity test procedure we have used an extra 2000 patterns per class for testing, thus 4000 patterns in total. We have again used the default learning scheme as depicted in Table 5-1.

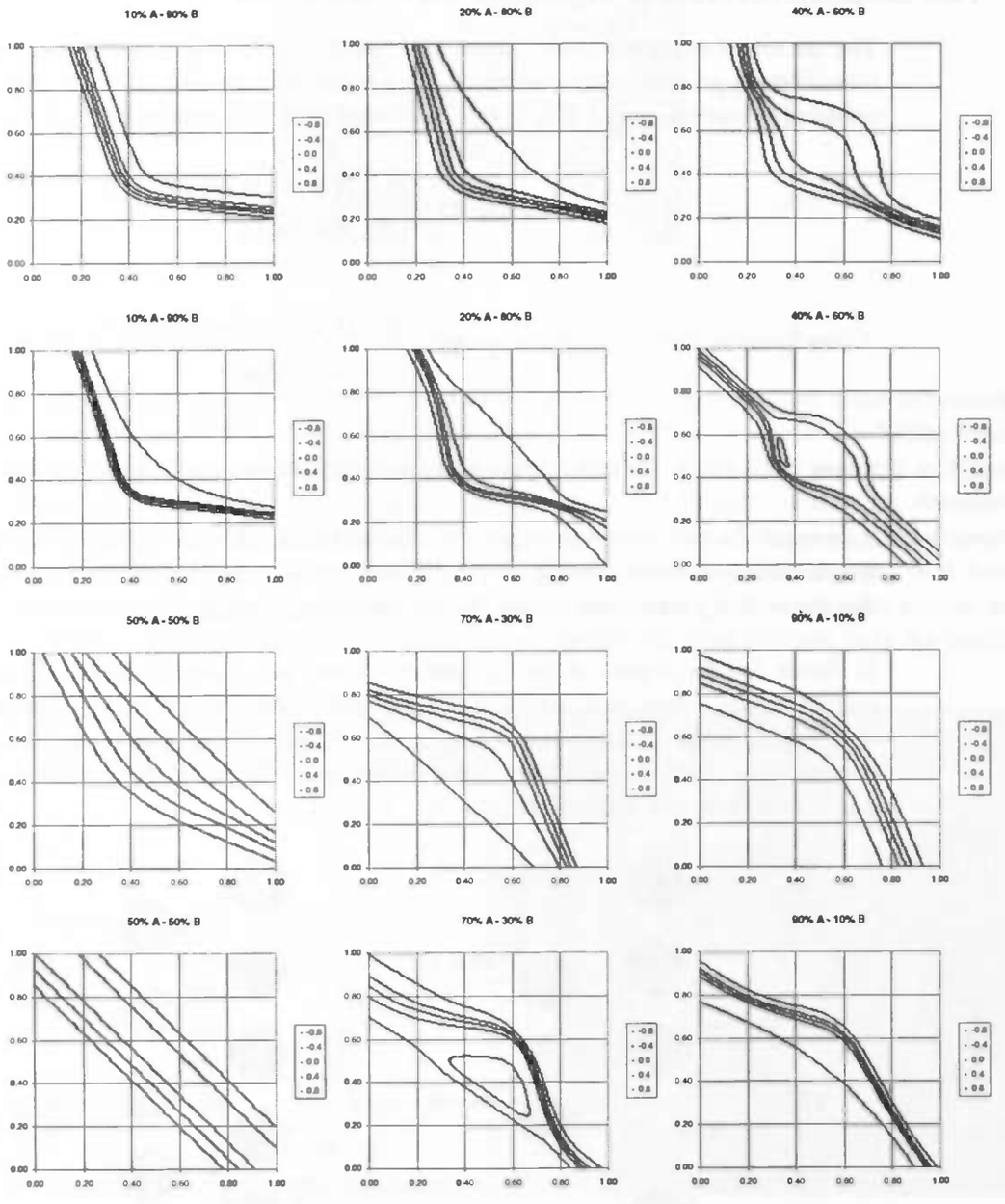


Figure 5.5: Each plot shows the differences in the two outputs of a classifier trained on classes A and B in the given ratio, corresponding to the input (both horizontal and vertical axis). Both classifier with one (odd rows) and with two (even rows) hidden layers appear to 'encapsulate' the class of which they have seen the most examples, which as a results get appointed only the smallest part of the input space. Also note that the classifier with two hidden layers seems to show some (minor?) signs of overfitting recognizable by the 'islands' in the plots.

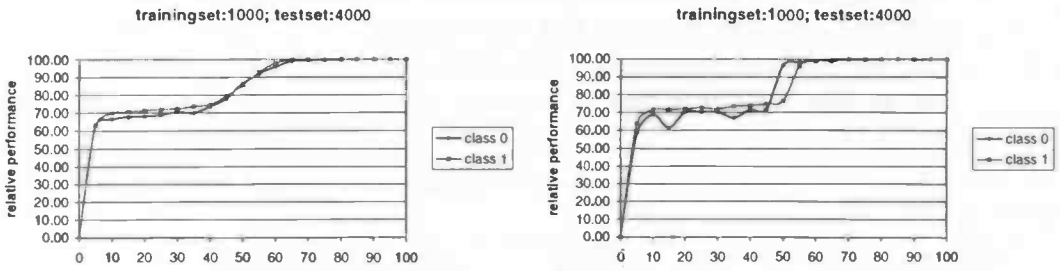


Figure 5.6: Results of the sensitivity test on the overlapping circles problem. The classifiers on the left side have a single hidden layer (10 neurons) and on the right side have two hidden layers (5,4 neurons).

Let us examine these plots. In Figure 5.5, when simply taking the higher score as the winner (no rejections) it becomes apparent that the classification result in the regions without overlap is pretty much independent of the used class ratio: in each plot the zero-difference line lies about somewhere over the region where the overlap occurs. As with our one-dimensional classification problem, the plots on the odd rows show a more grudual shift in the classification result when the class ratio in the training set changes than in the plots on the odd rows. These differences also show (more or less) in the Figure 5.6, showing the results of the sensitivity tests. The horizontal parts in these plots thus, again, seem to denote regions of little sensitivity to variations in the class ratios.

The result with the classifiers with two hidden layers shows, again, a somewhat more complicated pattern than the results from the classifiers with a single hidden layer. For instance: in the plots titled '40% A – 60% B' and '70% A – 30% B' some 'islands' seem to appear that do not show in the corresponding plots when training with a single hidden layer. (something similar also occurred in the experiments with the one-dimensional data set; take a close look at the plots titled '20% A – 80% B' in Figure 5.3). Again this seems to be due to the fact that a network with two hidden layers seems to be better capable of learning the actual class ratio at each input vector.

Note that the class of which the classifier has seen the largest number of examples (in contrast to what one might be inclined to think) gets appointed only the smallest portion of the input space. Or in other words, when the network is presented enough examples of a particular class, it seems to have a tendency of clearly marking out the space in which they occur. This might eventually lead to lousy extrapolation properties (for instance, if the data is more Gaussian distributed).

All in all the sensitivity plots in Figure 5.2 and in Figure 5.6 do give useful 'summaries' of output the corresponding plots like those given in Figure 5.3 and in Figure 5.5. Only some small peculiarities in the output plots do no show in the sensitivity plots (for instance the 'islands' in the plots when using two hidden layers). However, sensitivity plots are a lot easier to create and to interpret than output plots (especially when the number of input dimensions increases).

5.3 Stability on non-linear class boundaries without overlap

Next we would like to know if when we have a classification problem without overlap, that is, just any problem, then can we say at forehand that this will not lead to any stability problems (other than the ones described in Chapter 4). For this purpose we have generated a data set for a non-trivial, two-dimensional classification problem, as shown in Figure 5.7.

Both classes are symmetrical (with respect to each other) and have circular 'bulges' and matching 'indentations' such that they together can completely cover a rectangular area. Actually we have generated a small number of data sets with different horizontal distances between the two classes (in Figure 5.7 this distance is chosen to be equal to 0.25).

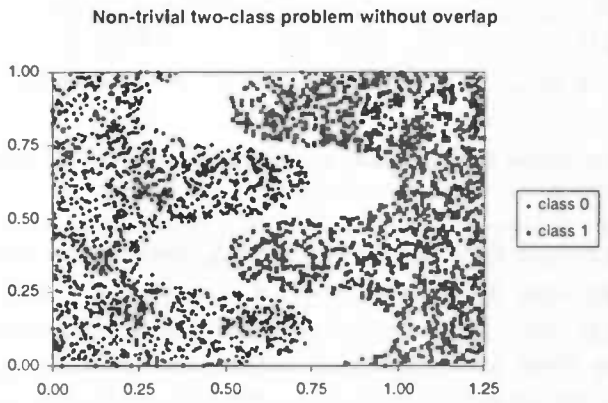


Figure 5.7: A scatterplot of a non-trivial, two-diminsional classification problem without overlap.

We have carried out experiments for the following values of the horizontal distance between the both classes: 0.0, 0.125 and 0.25. And again both classifiers with a single hidden layer (10 neurons) and classifiers with two hidden layers (5 and 4 neurons) as depicted in Figure 5.4 were used. We have used the training scheme as given in Table 5-2. Since the classifiers learned quite slowly we have used a lot training epochs this time. The training sets counted 1000 patterns each and the test set had 1500 patterns in total equally divided over both classes (thus, 750 patterns per class). See for the results Figure 5.8. Since the two classes are symmetrical and therefore similar results can be expected for both of them, only the results for class 0 are shown.

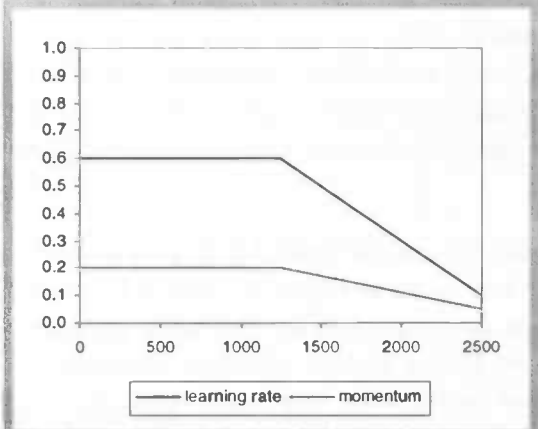
plot	data points		
	epoch:	learning rate:	momentum:
	0	0.6	0.2
	1250	0.6	0.2
	2500	0.1	0.05

Table 5-2: The training scheme used for the sensitivity tests with the classification problem with non-linear class boundaries without overlap.

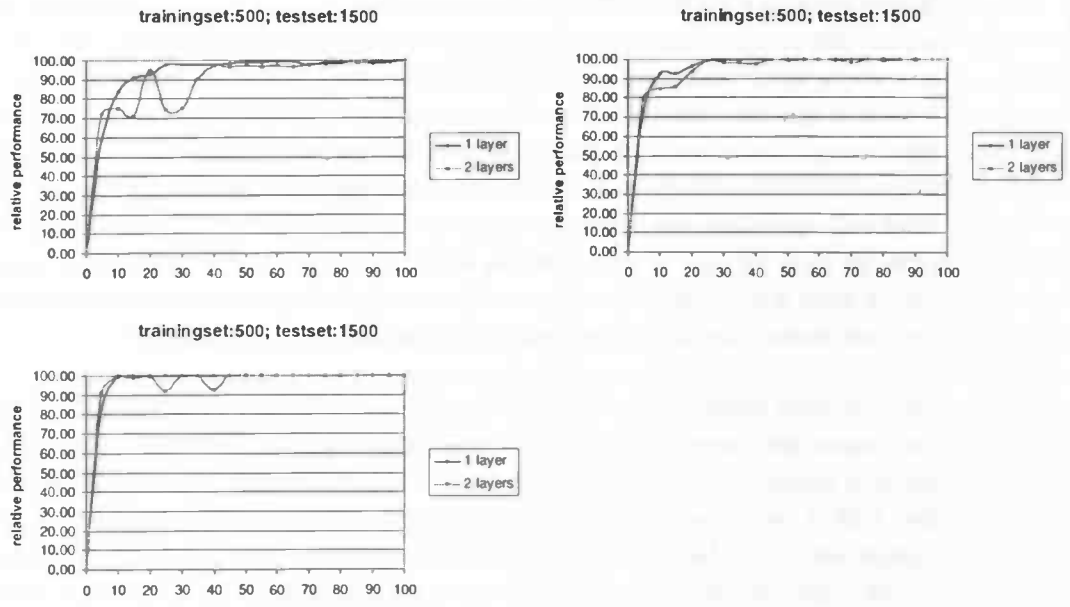


Figure 5.8: Results of the ratios test on the nontribial, nonoverlapping, two dimentional classification problem with the following values for the horizontal spacing between the two classes: 0.0 (upper left plot), 0.125 (upper right plot) and 0.25 (lower plot).

As we said, this seemed to be a rather difficult problem for our classifiers, especially with d chosen to be equal to 0.0. This can be seen in the upper left plot of Figure 5.8, where (for the classifier with one hidden layer) the line only starts to run flat at 25%, that is: the classifier seems to be insensitive to the class ratios as long as for both classes at least 25% of the patterns in the training set belongs to that class. As the parameter d is chosen larger, then only the region of insensitivity for class ratio grows. When looking at the plots for the classifier with two hidden layers, it appears that those results look a lot less stable, compared to results with the single-hidden-layered classifier. But note that being sensitive to class ratios does not necessarily mean that the overall performance will fluctuate just as much.

5.4 Effects of the size of the data set on class sensitivity

In this paragraph we will take a somewhat closer look at the effects that the sizes of the training and the test sets can have on the test results. We have therefore applied our procedure to both the overlapping circles data and the clouds database (see Appendix A for a description and Chapter 4 where we have carried out several experiments using this database). The results are shown in Figure 5.6 and in Figure 5.9. On the horizontal axis the percentage of patterns belonging to the class at hand is marked out.

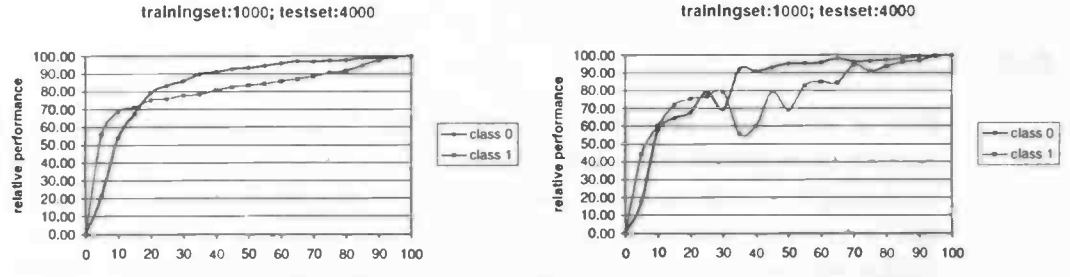


Figure 5.9: Results of the ratio test on the clouds database.

We have again used the training scheme as given in Table 5-1. The left plots are made using a classifier with a single hidden layer (containing 10 neurons) and for the plots on the right side a classifier with two hidden layers (containing 5 and 4 neurons) was used (see Figure 5.4 for graphical representations ofthese architectures). As the titles of the plots indicate, we have used training sets of 1000 patterns and a test set of 4000 unseen patterns, equally divided over the two classes. For each point in the plots only a single network was trained. This will mean that the actual performance can vary a few percents from the points in the plots (at least for plots using networks with a single hidden layer, see 4.3). Especially for the clouds data, it looks like classifiers with two hidden layers are more sensitive to variations in the class ratios than their counterparts with only a single hidden layer.

Then we have repeated the experiments with only 250 training samples and 750 test patterns. The results are shown in Figure 5.10 and Figure 5.11. Since only one classifier is trained per point, it was to be expected that the variation increases and the lines would be less smooth. But with a little imagination the plots on the left side (using the classifier with a single hidden layer) still look pretty much like the ones in Figure 5.6 and in Figure 5.9. The plots on the right side seem to have undergone a more serious metamorphose. These plots have actually become quite unusable. We can conclude from these results that when more data is available for training, classifiers tend to have less trouble with varying class ratios.

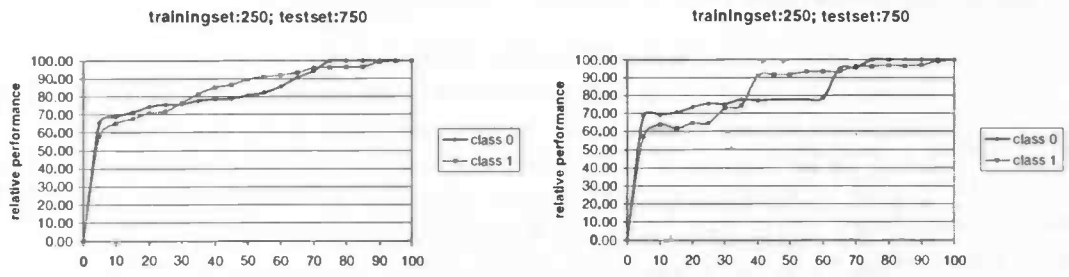


Figure 5.10: Results of the ratio test on the two overlapping circles problem.

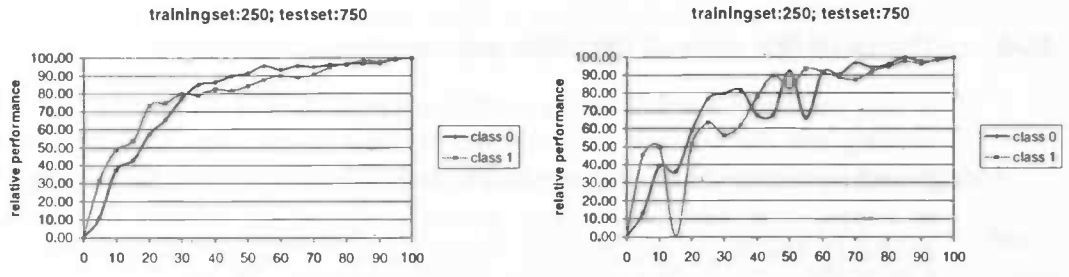


Figure 5.11: Results of the ratio test on the clouds database.

5.5 Conclusions

First of all, neural networks with two hidden layers tend to be much more sensitive to varying class probabilities than networks with only a single hidden layer. Only for the our most simple, one-dimensional classification problem it did not really show. The reason for this is probably that when working with neural networks of only a limited size (9 or 10 hidden neurons), neural networks with two hidden layers are better capable of doing that what an ideal classifier is thought to be capable of, namely: giving an "asymptotic approximation of the underlying a posteriori class probabilities" (see, for instance, Haykin,

1999). But this is, among other things, only possible when a sufficiently large amount of training samples is used (preferably an infinite amount). Let us now take another look at the plots in Figure 5.3 and in Figure 5.5. To obtain these results we have used only a limited amount of training samples, 1000 to be precise. We can see in these figures (perhaps it is most clearly in Figure 5.3) that the while plots obtained with the single hidden layer show smooth lines, the plots obtained with two hidden layers have more or less fitted the actual class probabilities at each input point. But (of course) they have not fitted the a posteriori class probabilities, but simply the ratios as they occurred in the training set. This can be concluded from the 'bulges' in Figure 5.3 and the 'islands' in Figure 5.5. In other words, classifiers with more than one hidden layer are more likely to cause overfitting problems. Note that the symptoms we have seen can hardly be classified as serious problems. Also keep in mind that even though we have seen some more serious stability problems on the clouds data (see Figure 5.11) this does not simply imply that the overall performance is fluctuating just as much. And so, it is perhaps not that surprising at all, that though a lot of attention is given to this problem called overfitting, it never really seemed to be a problem for our ('single-hidden-layered') networks. Perhaps using only a single hidden layer can be seen as a stabilizing technique in itself (even though no mention about it is made in any of the articles and books included in the References).

Second, when restricting ourselves to results obtained by networks with only a single hidden layer, it can be concluded that for classification problems without overlap, training on a 'incorrect' class ratios is no cause of stability problems. Only when one (ore more) class is (nearly) excluded from the training set one must be prepared for serious stability problems.

And finally, when the number of training samples is too small then even classifiers with only a single hidden layer start showing somewhat less stable results.

Chapter 6 Conclusions

When putting everything together we come to the following conclusions:

6.1 Limited amount of data

If we want to have an accurate estimate of the performance or of the stability of a classifier it is best to use an as large a independent test set as possible. This is because even a stable classifier may appear all but stable when a small test set is used to estimate its stability. In such a case, when only taking a single estimate of the performance of the classifier, we will most likely not be aware of the inaccuracy of this estimate. Therefore, when only a limited amount of data is available, it is simply impossible to tell whether any estimated instability is mostly due to the limited size of the training or due to the limited size of the test set. The only thing that can be relative accurately estimated when only a limited amount of data is given, is the average performance of the classifier. And thus, only when the estimated stability of a classifier is good enough there is no need to worry about any larger test sets.

6.2 Varying class ratios

Even if we think to have a stable classifier at hand we still might have a problem. For instance, when the classifier will be operating on data with (completely) different class ratios as the ones we have been training and testing our classifier with. If the classification problem is not without overlap, there is a major chance that this will affect the performance of the classifier. It therefore never hurts to check the class sensitivities over a wider range of input class ratios instead of only training on the given class ratios. Unfortunately there is not much that can be done if a certain classifier appears to be too sensitive for small changes in class probabilities. One of the best things to do is probably to make sure that you have the most accurate estimate of the a posteriori class probabilities and then train and test your networks according to these ratios.

6.3 Initial weight setting and order of presentation of patterns per epoch

Another thing about which not much can be done is that the effects of the random choice of initial weights and the random order in which the patterns are presented to the neural network classifier each epoch can have a more than negligible influence on the stability of the classifier. Depending on the 'hardness' of the problem at hand, this might result in maximum differences in performance up to 2.0 %. But unfortunately it is simply not feasible to select the most suitable settings at forehand.

6.4 Improving stability

And finally, there is one thing that can be done to improve the stability besides using more training data, applying a more suitable training scheme and that is choice of the architecture of the neural network classifier. According to our results, classifiers with more than one hidden layer are far more sensitive to variations in class ratios and show more tendencies to

overfitting. This does not mean that classifiers with more than one hidden layer should never be used for classification problems. For one thing, they seem to be much better in learning the actual class probabilities (when the training set is sufficiently large). But when you are having stability problems and you are using more than one hidden layer then it might be an idea to get rid of those extra hidden layers. Or, perhaps (as we have not actually tried it out), it might be an idea to apply any of the stabilizing techniques as described in Chapter 3.

6.5 Further research

In this report we have mostly limited our research to the effects of the size and composition of the available data set on the stability of any resulting neural network classifier. This is the main reason for instability as we have seen and unfortunately, besides using more data for training there is not much that can be done about this. But as we have also seen, some amount of instability stems from the choice of the training scheme and the chosen architecture. And since we have mostly tried to ignore their contribution to any measured instability it be worth a more thorough investigation. Also, in all of our experiments we have simply taken the highest output neuron as the winner. Yet it is quite common to use somewhat more sophisticated rules, including the use of a minimum threshold for a winner and a minimum difference in outputs for the winner and the runner-up. If there happens to be no absolute winner according to these rules, then the pattern will be rejected. That is, the classifier simply states that this particular pattern is to hard to correctly classify. It might be very well possible to construct neural network classifier that are far more stable while still attaining a reasonable performance if the proper rejection parameters are used. Though some research on these parameters has been done, so far it is not clear what parameters function best in what situation and what their effect of the stability is.

References

Atiya, A. and C. Ji, "How initial conditions affect generalization performance in large networks", IEEE transactions on neural networks, vol. 8, no. 2, p. 448-451, 1997.

Duda, R.O. and P.E. Hart, "Pattern classification and scene analysis", John Wiley & Sons, 1973.

Fukunaga, K., "Introduction to statistical pattern recognition", second edition, Academic Press, 1990.

Guyon, I., J. Makhoul, R. Schwartz and V. Vapnik, "What size test set gives good error rate estimates?", IEEE transactions on pattern analysis and machine intelligence, vol. 20, no. 1, p. 52-64, 1998.

Haykin, S., "Neural networks, A comprehensive foundation", second edition, Prentice-Hall International, 1999.

Hoekstra, A., R.P.W. Duin and M.A. Kraaijveld, "Neural networks applied to data analysis", Delft University of Technology, 1997.

Krogh, A., J.A. Hertz, "A simple weight decay can improve generalization", Advances in neural information processing systems 4, Morgan Kauffmann Publishers, San Mateo CA, p. 950-957, 1995.

Lawrence, S., I. Burns, A. Back, A. Chung Tsoi and C. Lee Giles, "Neural network classification and prior class propabilities", Tricks of the Trade, Lecture Notes in Computer Science State-of-the-Art Surveys, Springer Verlag, p. 299-314, 1998.

Maclin, R. and D. Opitz, "An empirical evaluation of bagging and boosting", The Fourteenth national conference on artificial intelligence, Province, Rhode Island, AAAI Press, 1997.

Reed, R., R.J. Marks II and S. Oh, "Similarities of error regularization, sigmoid gain scaling, target smoothing and training with jitter", IEEE transactions on neural networks, vol. 6, no. 3, p. 529-538, 1995.

Ripley, B.D., "Pattern recognition and neural networks", Cambridge University Press, 1996.

Sarle, W.S., "Stopped training and other remedies for overfitting", Precedings of the 27th symposium on the interface of computing science and statistics, p. 352-360, 1995.

Thimm, G. and E. Fiesler, "Pruning of neural networks", IDIAP, Research report, 1997.

Tsukuda, Y., H. Kurokawa and S. Mori, "Investigation of generalization ability by using noise to enhance MLP performance", Precedings of IEEE international conference on neural networks, vol. 5, p. 2795-2798, 1995.

Appendix A The clouds database

Aim

Study of the classifier behavior for heavy intersection of the class distributions and for high degree of nonlinearity of the class boundaries.

Description

The clouds data consists of a number of bidimentional distributions, divided in two classes. The data contains 5000 patterns, 2500 (50%) in each class.

Class 1 (C_1) contains the sum of three different gaussian distributions:

$$p(\mathbf{x} \mid C_1) = \frac{p_1(\mathbf{x} \mid C_1) + p_2(\mathbf{x} \mid C_1) + 2p_3(\mathbf{x} \mid C_1)}{4}$$

with

$$p_j(\mathbf{x} \mid C_1) = \frac{1}{2\pi} \frac{1}{s_{jx}} \frac{1}{s_{jy}} \exp\left(\frac{-(x - m_{jx})}{2s_{jx}^2} - \frac{(y - m_{jy})}{2s_{jy}^2} \right)$$

where m_{jx} and m_{jy} are the x and y means of distribution number j while s_{jx} and s_{jy} are their x and y standard deviations with their values given in the following table:

	s_{jx}	s_{jy}	m_{jx}	m_{jy}
$p_1(\mathbf{x} \mid C_1)$	0.2	0.2	0.0	0.0
$p_2(\mathbf{x} \mid C_1)$	0.2	0.2	0.0	2.0
$p_3(\mathbf{x} \mid C_1)$	0.2	1.0	2.0	1.0

Class 2 (C_2) contains a single normal distribution:

$$p(\mathbf{x} \mid C_2) = \frac{1}{4\pi} \exp\left(\frac{-\mathbf{x}^2}{2} \right)$$

A graphical representation of the two classes is given in Figure A.1 while a combined scatterpot of both classes is given in Figure A.2.

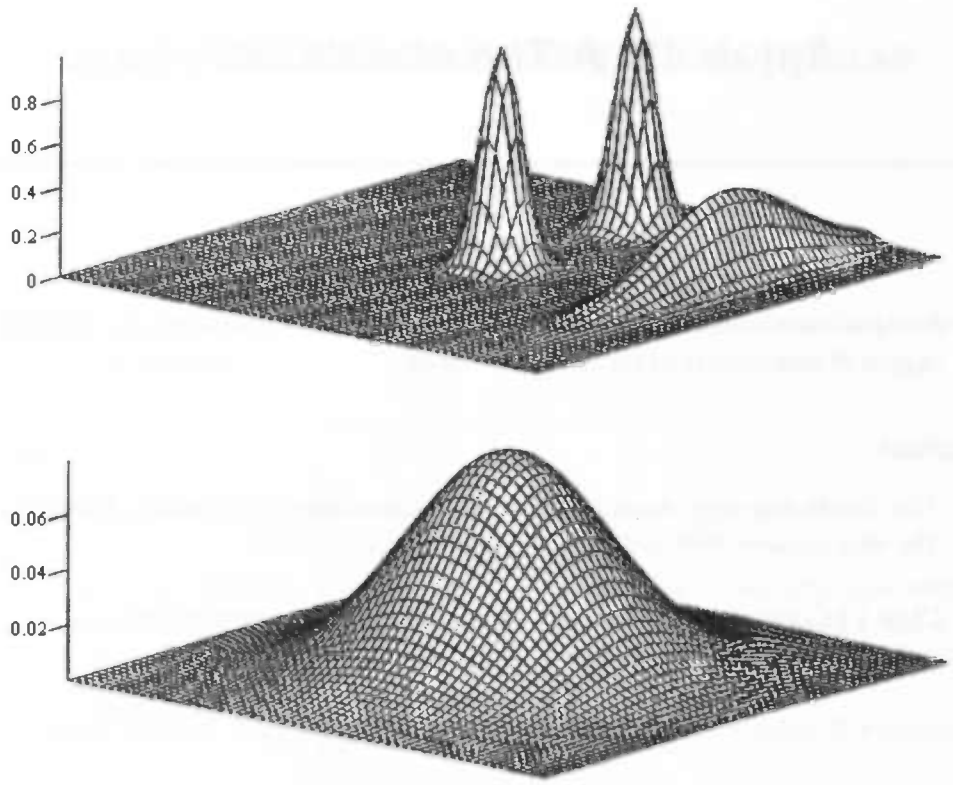


Figure A.1: A graphical representation of class 1 (above) and class 2 (below). The x- and y-axis both run from -3.0 to 3.0.

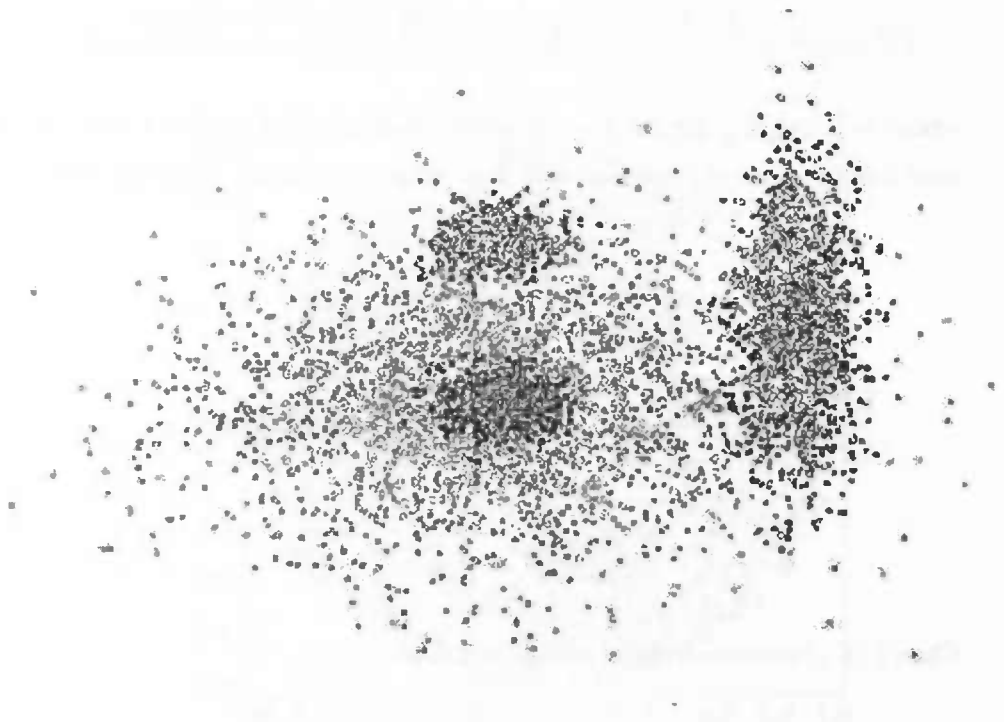


Figure A.2: A combined scatterplot of class 1 (dark) and class 2 (light).

Best theoretical Bayes confusion

The best theoretical Bayes confusion for this data set is given in the following confusion matrix:

Decision \ Class	1	2
1	94.63%	13.95%
2	5.37%	86.05%

This provides 9.66% for the average Bayes error (with 0.05% accuracy).

Appendix B The concentric database

Aim

Study of the linear separability of the classifier when some classes are nested in other without overlapping.

Description

Class 1 (C_1) contains the sum of three different gaussian distributions:

The concentric database consists of two bidimentional uniform concentric circular distributions, divided in two classes. The database is entirely contained in the square (0,0), (1,1). It contains 2500 instances, 921 in class 1 (36.8%) and 1579 in class 2 (63.2%).

The points of class 1 are uniformly distributed into a circle of radius 0.3 centered on (0.5, 0.5).

The points of class 2 are uniformly distributed into a ring centered on (0.5,0.5) with internal radius of 0.3 and external radius of 0.5.

A combined scatterplot of all the points of both classes is given in Figure B.1.

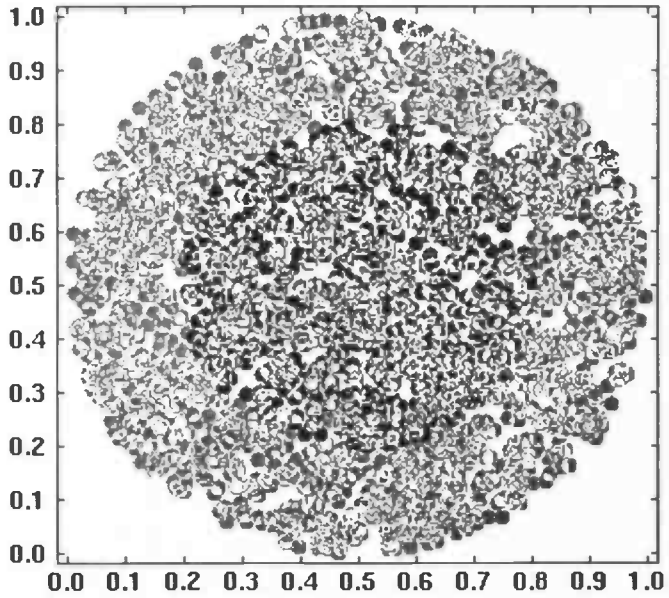


Figure B.1: A combined scatterplot of class 1 (dark) and class 2 (light).

Best theoretical Bayes confusion

Decision \ Class	1	2
1	100%	0%
2	0%	100%

(This is easy to see, since the classes don't overlap.)

Appendix C The number plates database

This database (named F02_40X3.pdb) contains 3143 alphanumeric characters, as they appear on number plates. Each picture of a character is reduced to 30 inputs by means of principal component analysis. Each character belongs to one of 36 classes.

The number of patterns is not uniformly distributed over the different classes. Some classes, like 'A', do not contain any patterns at all. Here is a list of all classes and the number of patterns belonging to it:

class '0':99	class '6':95	class 'C':0	class 'I':0	class 'O':0	class 'U':0
class '1':97	class '7':98	class 'D':122	class 'J':115	class 'P':136	class 'V':176
class '2':124	class '8':118	class 'E':0	class 'K':39	class 'Q':0	class 'W':0
class '3':98	class '9':104	class 'F':137	class 'L':109	class 'R':155	class 'X':121
class '4':92	class 'A':0	class 'G':111	class 'M':1	class 'S':129	class 'Y':46
class '5':125	class 'B':175	class 'H':136	class 'N':141	class 'T':118	class 'Z':126

The optimal Bayes error for this particular database is 0.0 (for the simple reason that a neural network classifier, trained on the entire database obtained a performance of 100%).