Ultrasonic Tissue Characterization Using Neural Networks

Maurice S. klein Gebbinck

November 4, 1992

Abstract

Ultrasonic tissue characterization is a technique where on the basis of parameters that have been acquired using ultrasound it is tried to derive some properties of the tissue under observation. At the Biophysics Laboratory of the Institute of Opthalmology at the Academic Hospital Nijmegen the applicability of this technique for the diagnosis of diffuse liver diseases is investigated.

An important component of a diagnostic system is the algorithm that classifies the patients into one of the to be discriminated groups. Uptil now discriminant analysis, which is a statistical method, has been used as classifier, but neural networks are known to perform this task also very well. In this thesis two types of neural networks, back-propagation and feature mapping, are investigated with regard to their classifying capabilities. It is shown that with back-propgagation better results can be achieved than with discriminant analysis.

One of the problems when dealing with neural networks is that the data set containing patients whose disease has already been diagnosed must be sufficiently large. Another topic of this thesis therefore is the generation of artificial data based on the original patients.

Contents

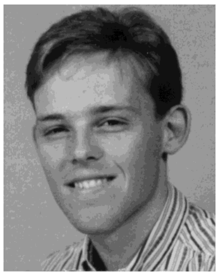
List of Tables

List of Figures

Preface

This report is the result of a project carried out in the period from September 1991 to October 1992 in order to obtain a Master's Degree in Computer Science. The project was a co-operation between the department of Real-Time Systems of the faculty of Mathematics and Computer Science at the University of Nijmegen and the Biophysics Laboratory of the Institute of Opthalmology at the Academic Hospital Nijmegen, under supervision of Theo Schouten and Johan Thijssen.

First of all I want to express my special thanks to Hans Verhoeven, Johan Thijssen and Theo Schouten, who were always willing to help me and were a source of inspiration. Furthermore I would like to thank Rien Cuypers for sharing his knowledge about SAS/STAT, Bernard Oosterveld for supplying the original patient data, Harry Duys for providing me with additional computational resources, Silvio Bierman for placing his Kohonen Neural Networks Simulator at my disposal, Parcival Willems for installing the software xgraph and gspreview, and Edwin klein Gebbinck for his constructive comments on this report. Last but not least I want to thank all members and students of the department of Real-Time Systems and the Biophysics Laboratory for the pleasant working atmosphere they created.



Maurice S. klein Gebbinck
Department of Real-Time Systems
Faculty of Mathematics and Computer Science
University of Nijmegen

Chapter 1

Introduction

This chapter provides a short introduction into the two main techniques on which this thesis is based. After that, the subject of the thesis is explained and its outline is given.

1.1 Why ultrasound?

Nowadays an clinician can apply several techniques to obtain information on a patient without having to operate on him. Each method has its pros and cons, some of which we will mention briefly.

The oldest and most wide-spread technique is *Tomography*[1]. Although the term may be unfamiliar nearly everybody has seen its product: X-ray photos. Tomography is very suited for detecting broken bones, but when dealing with abdominal injuries for instance it is useless. Another disadvantage is that it makes use of X-rays. This kind of radiation is harmful to biological tissues, therefore its application should be restricted.

A more recent method is *Computer Tomography* or CT[2]. This technique uses X-rays as well so, again, its use should be restricted. A big difference with ordinary tomography is that with CT many measurements are needed. All these measurements, or slices, are processed by a computer to get a three-dimensional model. With this model all sorts of useful operations can be performed. Some of them are obvious, like viewing from any angle or zooming in on certain areas. A more sophisticated possibility is the detection of tissues with different characteristics. This way a tumour may be detected, but peeling of the subject — omitting certain tissues, fat for example — is also possible.

The most recent technique is *Magnetic Resonance Imaging* or MRI[3]. With this method a model of the subject is constructed too, so all the possibilities mentioned in the paragraph on CT apply to MRI as well. A big difference with CT is that MRI uses electro-magnetic waves instead of X-rays. Today scientists assume that electro-magnetic waves are harmless to biological tissues.

[1] The term *Planigraphy* refers to the same method, but has become a bit out of date.

[2] Sometimes this technique is called *Computer Aided Tomography* or CAT.

[3] Other names for this method are *Nuclear Magnetic Resonance* or NMR, and *Magnetic Resonance* or MR.

Unfortunately MRI has an important disadvantage: the necessary equipment is very expensive. There are several reasons for the high costs. For instance many components are very complicated and must be made by hand. Secondly a lot of quality checking is necessary because a great precision is required. These factors limit the demand for such an apparatus, forcing the price up even more.

The method this thesis deals with is *Ultrasound Imaging*. This method has many advantages. First of all UI does not make use of any kind of ionizing radiation. Diagnostic ultrasound is harmless to biological tissues and can be applied unrestricted, even on pregnant women. Therefore it is often used to check on the status of an unborn child. Because ultrasound is completely different from ionizing radiation, the type of information obtained with it is very different as well. While radiation shows the presence of materials that absorb radiation (bone e.g.), ultrasound works best on soft tissues. The information obtained with UI therefore should be used complementary to the information obtained by other techniques. Another advantage of this method is that it is relatively cheap. The reason for this is that the equipment is not extremely complicated and the components themselves are not very expensive. Finally the technique is very fast, making it possible to monitor moving subjects like the heart. A term that is often used in this context is real-time imaging. All these factors have made that ultrasound imaging takes a firm position among other medical imaging techniques.

Unfortunately the method has a big disadvantage as well. The quality of the images is a lot poorer than of those obtained by CT or MRI. Today there are several approaches to improve the quality. A lot of research is done to improve the equipment, for equipment with a higher resolution improves the images. Other research is focussed on the underlying physics. A better understanding of the imaging process may lead to the correction for undesired phenomena.

1.2 Why neural networks?

The human brain with its remarkable features has always fascinated scientists. The last few decades it is tried to simulate the workings of the brain using a simple model, called a *neural network*. Today, simulating the brain is not the only goal anymore, for research has shown that neural networks can be used as a new computational model, performing at some tasks much better than other computational models.

A neural network has many advantages over traditional techniques. First of all the development of software is greatly reduced, compared to the traditional way of solving a problem. While traditionally for every new problem that was to be tackled a new algorithm had to be developed and programmed, for neural networks only the software for the simulator has to be written. This is a big advantage, because not only can the algorithm be very complex and thus difficult to find, the development of software that does not contain too many errors costs a lot of time and money as well. All this does not apply to neural networks, for a neural network is trained by continuously presenting a problem together with its solution. This way problems that are somewhat vague can be solved also. However some knowledge about the problem domain may be required to get a good performance of the neural network.

In a neural network information is encoded in a distributed and possibly redundant fashion. An advantage of such a storage scheme is that the network can undergo partial destruction and will still be able to function correctly. This is especially handy for use in hostile environments, like space, or when correct functioning of the application is of the utmost importance (defense systems e.g.).

Another advantage of a neural network is that the input presented to it is allowed to be incomplete or incorrect to some extent. Traditional systems however would crash if not all input is given.

Finally neural networks can well be parallelized. This makes the application of parallel computers, which are much cheaper than other computers with the same computational power, possible.

As said before, neural networks learn by example. Although this has many advantages, it is at the same time the cause for a disadvantage: neural networks need many examples to get a good understanding of a problem and its solution. If the number of examples is not sufficient, generation of additional samples can be a solution.

The most important disadvantage of neural networks is that the theoretical background is very limited. The behavior of a neural network can not be predicted yet. This makes it difficult to know if a problem can be tackled, and, if it can indeed be tackled, how the network must be tuned to get the best performance. It is for this reason that neural networks usually involve a lot of experiments. However some problems, like the mapping, recognition and completion of patterns, are now known to be well suited to be solved with a neural network. Other tasks, administration for example, can better be solved using traditional techniques.

1.3 On this thesis

At the Biophysics Laboratory of the Institute of Opthalmology at the Academic Hospital Nijmegen a different approach is taken to improve the applicability of ultrasound imaging. Their line of research is that of the *ultrasonic tissue characterization*. From several features derived from the ultrasound signals it is tried to determine the disease a patient suffers from. These features are of two types: acoustical parameters, for instance the attenuation, and parameters determined from the ultrasound image like the average gray-level. The philosophy behind this approach is that some diseases alter the structure and properties of certain tissues, resulting in a different response to the exposure with ultrasound.

Uptil now the classification of tissues based on these features is done using discriminant analysis. With this method good results are obtained [1], but neural networks might perform even better, since neural networks are not limited to linear or quadratic separation functions as discriminant analysis is. Just like in other fields, researchers of UI investigate the applicability of neural networks as well [2, 3, 4, 5]. For tissue characterization neural networks seem to be useful, both in the field of UI [5] and in other comparable fields [6, 7].

In this light, this graduation-assignment is to investigate the use of neural networks for ultrasonic tissue characterization of diffuse liver diseases. The results are to be compared with the results obtained with discriminant analysis, in order to see which of the methods prevails.

Apart from this chapter, which was intended as a short introduction, this thesis deals with the following subjects. In chapter 2 the fundamentals of ultrasound imaging are explained. Some clues on how tissue can be characterized using UI are given as well. Chapter 3 discusses the problem domain and the operations that must be performed on the raw data to get it in the right format. Discriminant analysis and the results obtained with it are dealt with in chapter 4. In chapter 5 back-propagation is explained and the results of this method are presented. Feature mapping and its performance are the subject of chapter 6. Finally in chapter 7 the results of the three methods are compared and conclusions are drawn. Furthermore some suggestions for future research are presented.

Chapter 2

Ultrasound imaging

This chapter is about the fundamentals of ultrasound imaging. First of all, it is explained what ultrasound is and how it can be generated. Next the interactions of ultrasound with the medium it travels through are dealt with. Finally it is shown how the acquired data can be transformed into an image. See [8] for more detailed information on this subject.

2.1 The transducer

Sound with frequencies so high that humans can not hear it, is called ultrasound. Usually this implies a frequency of 20 kHz or more. Unlike most other waves, sound waves are longitudinal. This means that the direction in which the particles that propagate the wave move, is parallel to the direction of the wave. Such a wave is generated by a *transducer*.

A transducer is a piece of material that exhibits the *piëzo-electrical effect*[1]. This means that if a voltage is applied on the material, the volume of that material changes, generating a sound wave. The effect is the strongest when the frequency of the alternating voltage is equal to the resonance frequency of the transducer[2]. If for some reason a different frequency is needed, a different transducer must be used.

The piëzo-electrical effect also entails the opposite phenomenon. If the volume of the material changes, for example due to an incoming sound wave, a voltage is generated. A transducer can therefore act as a receiver of ultrasound as well.

2.2 The pulse-echo method

As will be explained in section 2.3, ultrasound travelling through a non-homogeneous medium will create echoes. These echoes can be received by the same

[1]These materials, such as some ceramics and crystals, consist of dipoles that have been polarized, i.e. forced to point in the same direction.

[2]The resonance frequency depends on the thickness of the transducer; the thicker the piece of piëzo material, the lower its resonance frequency.

material	ρ (kgm^{-3})	c (ms^{-1})	Z (Rayl)
muscle	$1.07\ 10^3$	$1.56\ 10^3$	1.67
fat	$0.95\ 10^3$	$1.45\ 10^3$	1.38
kidney	$1.04\ 10^3$	$1.56\ 10^3$	1.62
bone	$1.91\ 10^3$	$4.08\ 10^3$	7.80
gas	1.30	$0.34\ 10^3$	$4.42\ 10^{-4}$
water	$1.00\ 10^3$	$1.47\ 10^3$	1.47

Table 2.1: Acoustical impedance for several materials.

transducer that generated the ultrasonic wave. This way of scanning a subject is called the *pulse-echo method*.

First a short electrical voltage is set on the transducer, resulting in the transmission of ultrasound. The transducer is heavily damped to limit the length of the ultrasonic pulse, which is usually only 2 to 5 periods. The ultrasonic wave consists of a band of frequencies centered around the resonance frequency of the transducer. As soon as the transducer is in rest again it can be used to transform the ultrasonic echoes into an electrical signal, called the *Radio Frequent signal* or RF-signal. From the RF-signal a lot of useful information can be extracted.

2.3 Physical interactions

To understand the data acquired with ultrasound imaging, it is necessary to understand the interactions between ultrasound and the medium it travels through. In this section these interactions will be briefly discussed.

2.3.1 Reflection

Biological tissues[3] contain many inhomogeneities without which ultrasound imaging would be impossible. An inhomogeneity is a particle with a different acoustical impedance than the surrounding tissue.The acoustical impedance (Z) depends on the density (ρ) of and the speed of sound (c) in the tissue in the following manner:

$$Z = \rho c \tag{2.1}$$

In table 2.1 the acoustical impedance for several tissues can be found.

If ultrasound hits an inhomogeneity bigger than its wave-length, part of it is reflected. Such an inhomogeneity is called a *reflector*. Some examples of reflectors are blood-vessels, organs, tumours etc. The bigger the difference in acoustical impedance, the bigger the part that is reflected. In table 2.1 it can be seen that the encounter of gas[4] or bone results in high reflection when travelling through biological tissue. The occurrence of shadow — the ultrasound is totally

[3] From here on the term biological tissues instead of medium will be used, because that is in my case what the medium consists of. Other substances in the human body, like bone, water, gas etc., are considered to fall under this term as well.

[4] This is why gel is used between the transducer and the skin.

reflected and doesn't reach the tissue behind a reflector anymore — is a real possibility.

The depth of a reflector (z) can be determined from the RF-signal. The following simple proposition can be applied:

$$z = \frac{ct}{2} \tag{2.2}$$

Here c represents the speed of sound in the tissue under observation, and t equals the time elapsed between the transmission of the ultrasonic wave and the reception of its echo. It is obvious that reflection is dependent on the positions of the reflectors, i.e. the depth.

That part of the ultrasound that is not reflected, travels on. The direction in which it travels will be slightly different from its original direction. This phenomenon, well-known in optics, is called *refraction*. According to the law of Snellius applied to ultrasound, the amount of refraction depends on the difference in the speed of sound between the two tissues. High refraction leads to a wrong image of a subject, because the reflectors aren't depicted on their right place anymore. It is possible that the difference in the speed of sound between two tissues is so high that instead of refraction total reflection occurs being another cause of shadow. In table 2.1 you can see that gas and bone result in high refraction when travelling through biological tissue.

2.3.2 Scattering

An inhomogeneity that is small compared to the wave-length of the ultrasound used, is called a *scatterer*. If ultrasound hits a scatterer, the ultrasonic wave is scattered in all directions. Usually the scatterers are uniformly distributed over the tissue, so scattering doesn't depend on the depth. Scattering however is dependent on the type of tissue, for different tissues contain scatterers which are different in size and density. This is one of the phenomenona that makes ultrasonic tissue characterization possible. Another factor that determines the amount of scattering is the frequency of the ultrasound. This dependency is of exponential kind with the exponent between zero, for scatterers with a size in the order of the wave-length, and four, for scatterers that are much smaller.

Scattering has a huge impact on so called B-mode images. In section 2.5 this will be explained.

2.3.3 Absorption

Part of the energy of the ultrasonic wave that travels through biological tissues is transformed into heat. In the early days of ultrasound imaging scientists assumed this was caused by friction, but today it is commonly accepted that relaxation processes are the main reason. When ultrasound travels through a biological tissue, many changes in volume and pressure take place. If all these changes are in phase with each other, no energy will be lost. Generally however this will not be the case, due to the already mentioned relaxation processes.

Because elasticity of the medium is an important factor, absorption is dependent on the type of tissue. This is a second phenomenon that can be of use for ultrasonic tissue characterization. Another factor is the frequency of the ultrasound. The amount of absorbtion is proportional to the frequency to the power 1.5. As the ultrasound travels through the tissue and is absorbed by it, the central frequency of the ultrasound will be shifted downward.

Absorption accounts for approximately 90% of the attenuation of an ultrasonic wave, while scattering accounts for the rest. If the frequency of the ultrasound used is increased, scattering accounts for a bigger part of the attenuation.

The amplitude spectrum (P_z) at a certain depth z from the transducer is described by

$$P_z(f) = P_0 e^{-\alpha(f)z} \tag{2.3}$$

with P_0 representing the initial amplitude spectrum of the ultrasonic wave and $\alpha(f)$ representing the attenuation-coefficient of the tissue. Using this equation the amount of attenuation can be determined experimentally, so the RF-signal can be corrected for it. Much more on this subject can be found in [9].

2.3.4 Diffraction

When you examine the beam of ultrasound produced by a transducer regardless of the tissue it travels through, you find that it isn't constant of intensity nor of shape. The beam can be divided in two different zones: the *Fresnel zone* and the *Fraunhofer zone* (see figure 2.1). In the Fresnel zone, or near field, the beam is constant of shape, but it differs greatly in intensity due to interference. In the Fraunhofer zone, or far field, the beam doesn't suffer from complete extinction, but diverges slowly. These effects are the result of the finiteness of the transducer. The ultrasonic wave that reaches a certain point in front of the transducer is composed of contributions coming from all points on the transducer surface. If the difference in distance between two such contributions is half a wave-length of the ultrasound used, they will extinct each other. The Fresnel zone is defined to be that part of the ultrasonic beam where the maximum difference in distance between a point in this zone and a point on the transducer surface is greater than or equal to half the wave-length. In the Fraunhofer zone the difference in distance is always smaller than half the wave-length, so extinction is impossible. If the transducer would be infinite, all contributions that reach a certain point would be exactly the same as the contributions that reach another point. This would result in a beam that is constant of intensity and of shape.

As will be explained in section 2.4.1, it is sometimes necessary to focus the ultrasonic beam. Focussing influences the shape of the beam and increases diffraction. But just like the factors mentioned in the previous paragraph — the frequency, which depends on the thickness of the transducer, and the size of the transducer surface — focussing is dependent only on the system, not on the tissue. The RF-signal can be corrected for the diffraction effects. A description of this method can be found in [10].

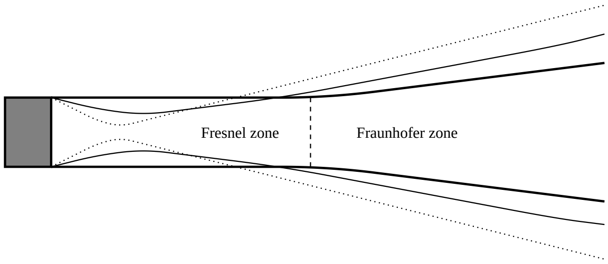


Figure 2.1: Shape of the ultrasonic beam; the thick solid line represents the unfocussed beam, the thin solid line the beam with medium focus and the dotted line the beam with short focus.

2.4 Construction of images

Many different techniques exist to graphically represent echo signals. Besides A-mode and B-mode images, which we will explain in more detail in sections 2.4.2 and 2.4.3, C-mode, AB-mode and M-mode images exist. They all serve different purposes. C-mode images for instance are useful for a three-dimensional view on a subject, while M-mode images can be used when monitoring moving structures like the heart. The quality of an image depends of the resolution.

2.4.1 Resolution

There are two types of resolution: *axial* and *lateral* resolution. Good values for both are necessary for a good image.

The axial resolution is defined as the resolution in the direction parallel to the axis of the transducer. It is a measure for the minimum distance two reflectors that are situated in the ultrasonic beam must have in axial direction to produce two detectable echoes. This means that the amplitude between the two peaks must have been decreased with at least 50%. If the two reflectors differ less in depth, the wave-front already reaches the second reflector, while the first is still reflecting the rest of the wave. These echoes will interfere so that it looks as if only one reflector is present. It is obvious that the axial resolution depends on the length of the ultrasonic pulse: the shorter the pulse, the better the resolution. The length of the pulse is defined by the number of periods the pulse consists of, and the frequency of the ultrasound used. Usually the number of periods is already very small, so the only way to improve the resolution is increasing the frequency. Unfortunately this will increase the attenuation as well, resulting in a much smaller penetration depth. Table 2.2 shows that in some fields of medicine the region of interest isn't situated very deeply, so a good axial resolution can be achieved.

The lateral resolution is the resolution in the direction perpendicular to the axis of the transducer. It defines how far two reflectors that are at equal depth must be apart in lateral direction so that they produce two detectable echoes. This resolution depends on the diameter of the ultrasonic beam: the smaller the diameter, the better the resolution. Due to diffraction the resolution is dependent of the depth. A way to improve the resolution at a certain depth is

field	frequency (MHz)	resolution (mm)	depth (cm)
brain	1-2	2.5	33
heart	2-5	1.1	14
deep organs	2-5	1.1	14
breast	3-5	1.0	13
pediatrics	2-7	0.8	11
superficial organs	5-10	0.5	7
opthalmology	7-20	0.3	4

Table 2.2: Axial resolution and penetration depth

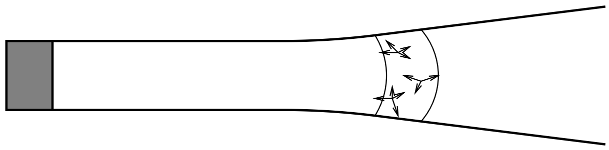


Figure 2.2: A sample resolution-volume containing several scatterers.

focussing, see figure 2.1. Usually focussing is achieved by bending the transducer surface or by using an acoustical lens, but a phased-array transducer will do the trick as well. Unfortunately focussing amplifies the diffraction effect, making the beam much wider after the focal point. This forces the user to select a region of interest to focus on.

A term that is often used in ultrasound imaging is *resolution-volume*, an example of which can be found in figure 2.2. Echoes occurring at a certain point of time can only be caused by inhomogeneities present in the according resolution-volume. As was to be expected resolution-volumes depend on length of the ultrasonic pulse, the size of the transducer surface and the speed of sound in the tissue under observation. Focussing alters the shape of the volume as well.

2.4.2 A-mode images

The most simple representation of RF-signals is the *A-mode* imaging technique in which the A stands for amplitude. It is an one-dimensional representation of the amplitude of the echoes as a function of the depth. Before you get such an image, the RF-signal must undergo several operations.

First of all the signal is amplified, which is necessary because the ultrasonic pulse contains little energy to avoid possible damage to the tissue. Usually this amplification is non-linear, because by using logarithmic amplification you get a more dynamic image: the ratio between the minimum and maximum signal is increased. Sometimes this is called compression. Often this amplification is combined with Time Gain Compensation or TGC. TGC is a time dependent amplification of the signal to correct for attenuation. Without TGC the tissues at greater depth wouldn't be clearly depicted.

The second step is demodulation of the signal. The type of demodulation used for the creation of A-mode images determines the amplitude from an echo

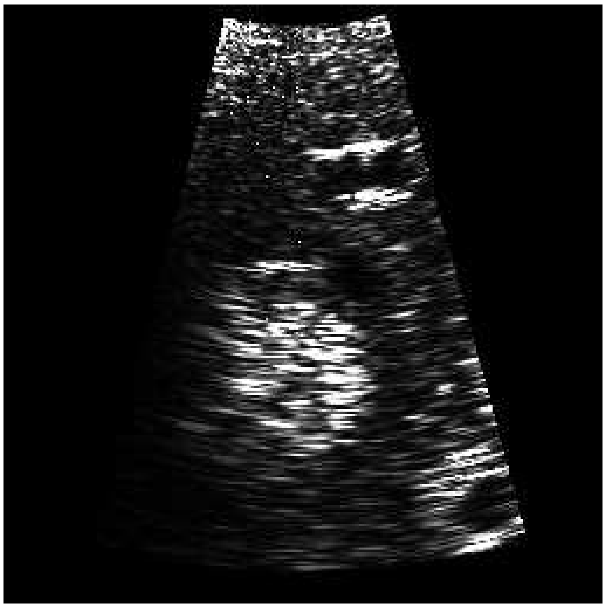


Figure 2.3: B-mode image of a liver displaying a tumour (middle) and two blood vessels (upper and lower right); the entire image is covered with speckle.

signal. Other information of the signal, like the phase, is discarded.

Filtering is the last step in the pipe-line. High frequencies are filtered out of the signal to smoothen it. One must be careful what frequencies to filter out, because the resolution of the image must not be affected.

2.4.3 B-mode images

By translating the transducer in lateral direction, you get a series of A-mode images. From these images a two-dimensional image can be constructed. A *B-mode* image is nothing more than the successive depiction of this series with the amplitude represented by brightness. An example of a B-mode image can be found in figure 2.3.

Because B-mode images contain a lot of information, organs can be examined for position and size, the evolution of an unborn child can be checked etc. All this is the result of reflection occurring at the boundary between an organ e.g. and its surrounding tissue. But these reflections are not the only things that can be seen in B-mode images. Tissues that are expected to be homogeneous are covered with some kind of texture, which is called *speckle*, resulting from the scattering in that tissue. Most people consider these echoes as noise that should be removed, but it is possible to characterize the tissue by it.

2.5 Speckle

Speckle is a phenomenon that was first described in connection with lasers. A good explanation of this can be found in several articles by Goodman [11, 12].

In ultrasound imaging speckle is caused by scatterers. When an ultrasonic wave hits a scatterer, the ultrasound is scattered is in all directions. Some of the ultrasound is scattered back to the transducer. At the transducer surface waves coming from several scatterers in the same resolution-volume are received at the same time, see figure 2.2. Because the scatterers all have a different distance to the transducer surface, the waves differ in phase and amplitude. This way the interfering waves may extinct or amplify each other.

As the resolution-volume changes to a greater depth or the transducer is translated, a completely different interference occurs. This way an irregular pattern of echoes is created. As speckle is intrinsic to UI, it can not be removed by increasing the power of the ultrasound. When regarding speckle as noise, it is of multiplicative instead of additive nature.

Speckle is described using first- and second-order statistics. First-order statistics include the mean amplitude and the signal-to-noise ratio. The second-order statistics include the auto-covariance in both axial and lateral direction. These statistics are dependent on the imaging system and the density of the scatterers. If the density is above a certain level speckle only depends on the imaging system. This situation is often referred to as *fully developed speckle*. For fully developed speckle a model can be constructed [13]. In sections 2.5.1 to 2.5.4 theoretical and practical results [14, 15] are compared.

2.5.1 Mean amplitude

Theoretically the probability distribution function for the magnitude of the amplitude can be given by

$$p(A) = \frac{A}{\sigma^2} e^{\frac{-A^2}{2\sigma^2}} \quad (A > 0) \tag{2.4}$$

with σ^2 the backscattered signal power. This probability distribution function is known as the Rayleigh distribution function. For this function it is known that

$$\begin{aligned} \mu_A^2 &= \frac{\sigma^2 \pi}{2} \\ \sigma_A^2 &= \sigma^2 (2 - \frac{\pi}{2}) \end{aligned} \tag{2.5}$$

where μ_A stands for the mean amplitude and σ_A represents the standard deviation of the amplitude.

The received signal power depends on a lot of factors. First of all it is dependent on the power of the transmitted ultrasound, for an increase of this power results in an increase of the backscattered signal power. This is meant with the remark that speckle is noise of multiplicative nature. Secondly the signal power is proportional to the number of scatterers. Due to diffraction the diameter of the ultrasonic beam is at its smallest in the focal zone, so the number of scatterers here is smaller than in any other resolution-volume. However, the energy of the transmitted ultrasonic wave per volume unit is much higher. Altogether this results in a higher received signal power. Equation 2.5 shows that the mean amplitude is proportional to the square root of the signal power, so the mean amplitude will be dependent on these factors as well.

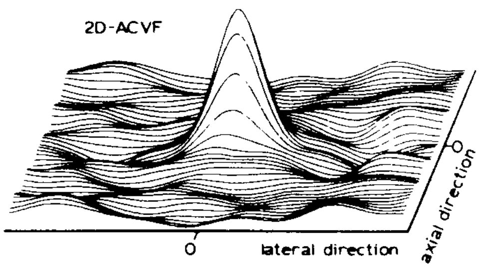


Figure 2.4: The auto-covariance function.

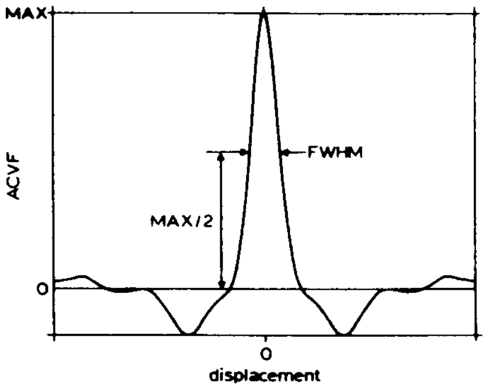


Figure 2.5: The full width at half maximum.

2.5.2 Signal-to-noise ratio

The signal-to-noise ratio or *snr* is defined as

$$snr = \frac{\mu_A}{\sigma_A} \tag{2.6}$$

From equation 2.5 it can be calculated that the *snr* will have a value of 1.91 for fully developed speckle. If the speckle is not fully developed — the density of the scatterers is below a certain level — the *snr* will be lower. In contrast to the mean amplitude, the *snr* isn't very dependent on the depth. This makes it suitable for ultrasonic tissue characterization.

2.5.3 Axial auto-covariance

Second-order statistics are used to describe the two-dimensional properties of speckle. The two-dimensional auto-covariance function (ACVF) is a measure of the average size of speckle in a B-mode image. It is defined as

$$ACVF(x,z) = \int\int \{A(x',z') - \mu_A\}\{A(x'+x, z'+z) - \mu_A\}\ dx'dz' \tag{2.7}$$

with A(x,z) the amplitude at position (x,z). An example of the ACVF is displayed in figure 2.4. The ACVF is characterized by the full width at half maximum (FWHM) in both axial and lateral direction, see figure 2.5. For fully developed speckle in the focal zone it can theoretically be derived that in axial direction

$$FWHM_{ax} = \frac{0.26}{\sigma_f} \tag{2.8}$$

with σ_f the standard deviation of ultrasonic pulse in the frequency domain in MHz.

The only factor that seems to influence the axial FWHM is the density of the scatters. Unfortunately the limit defined by equation 2.8 is already reached with relatively small densities. This disqualifies the axial FWHM as a good parameter for ultrasonic tissue characterization.

2.5.4 Lateral auto-covariance

As mentioned in section 2.5.3 the lateral auto-covariance can be characterized by the full width at half maximum in lateral direction. For fully developed speckle in the focal zone it can be shown that in lateral direction the following equation holds:

$$FWHM_{lat} = 0.864 \frac{\lambda F}{D} \tag{2.9}$$

Here λ represents the wave-length, F the focal distance and D the diameter of the transducer.

The lateral FWHM increases dramatically with increasing depth. This is mainly due to frequency dependent attenuation, which results in a lower central frequency at greater depths. As was predicted by equation 2.9, this will increase the FWHM. Another factor the FWHM in lateral direction depends on is the scatter density. The FWHM decreases approximately proportional to the logarithm of the density. Fortunately the limit defined by equation 2.9 is reached only at relatively high densities. If a correction is made for the frequency dependent attenuation, this parameter can be useful for the characterization of tissues.

Chapter 3

The data set

In this chapter we will explain what the data exactly represents. Furthermore, two methods for the generation of additional data are explained. Finally a description is given of the operations the data has to undergo before it can be presented to a neural network.

3.1 Definition of the classes

At the Biophysics Laboratory a database is maintained, containing patients suspected of suffering from some sort of diffuse liver disease. For each patient his number, age, kind of disease and the values of 26 parameters obtained with ultrasound imaging are recorded. What disease the patient suffered from is determined through a biopsy. The four classes with the largest number of patients have been selected in addition to a class consisting of people with a healthy liver. The five groups with their characteristics are the following:

A: people with a healthy liver; their classification in the database is 1;

B: people suffering from primary biliary cirrhosis; their classification in the database is 28, 281 or 282;

C: people suffering from hepatitis/cirrhosis; their classification in the database is 22, 23, 24, 26 or 27;

D: people suffering from acute hepatitis; their classification in the database is 20 or 21;

E: people suffering from alcoholic hepatitis/cirrhosis; their classification in the database is 25.

The above classification will be referred to throughout the whole thesis.

3.2 Description of the parameters

The database contains the values of 26 parameters per patient obtained by ultrasound imaging, from which five have been selected that will be used for ultrasonic tissue characterization in this thesis. The selection was based on the results of earlier research, partially described in [1]. In the remainder of this section these parameters, ordered in decreasing discriminating ability, are briefly reviewed.

Mean amplitude

This parameter has already been explained in section 2.5.1. The most important property of this parameter is that it is proportional to the square root of the backscattered signal power, which in its turn depends on the number of scatterers. The symbol for the mean amplitude in this thesis is μ_A.

Attenuation-coefficient beta

As mentioned in section 2.3.3 attenuation in relatively homogeneous tissues is caused by absorption and scattering. Because these two factors are dependent on the tissue, the attenuation is a good parameter for the characterization of tissues.

The frequency dependent attenuation can be described by the attenuation-coefficient $\alpha(f)$. A method for determining this coefficient can be found in [9]. Usually it is the sum of a number of terms of which the higher order terms are negligible small at relatively low frequencies. A good approximation therefore is

$$\alpha(f) = \beta f \tag{3.1}$$

in other words a linear fit through the origin. In this thesis the parameter will be referred to as β.

Signal-to-noise ratio

In section 2.5.2 more about the signal-to-noise ratio can be found. The *snr* is dependent on the density of the scatterers. If the density is higher than a certain level, the *snr* has a value of 1.91. For tissues with such high densities the *snr* is useless.

Backscatter-coefficient

It is already mentioned several times that tissue can be characterized by its scattering properties. One of the factors scattering depends on is the size of the scatterers, for which a good approximation can be determined from the backscatter spectrum.

First of all the backscatter spectrum is dependent on the spectrum of the ultrasonic wave, usually being a narrow band of frequencies. But due to the size of the scatterers, some frequencies in this band are scattered back more than others. If you make a linear fit of this spectrum, not necessarily through the origin, the slope appears to be a good measure for the size. The symbol for the slope in this thesis is S.

Attenuation-coefficient alpha

Just as β is a good approximation of the attenuation-coefficient, there are many more good approximations possible. One of them is the linear fit not necessarily through the origin, described by

$$\alpha(f) = \alpha_0 + \alpha_1 f \tag{3.2}$$

The slope α_1 appears to be a good parameter for ultrasonic tissue characterization.

3.3 Some statistics

To get a better insight into the data some statistics are presented in tables 3.1 to 3.5. From these tables it can be concluded that separation of the different classes is not trivial, maybe even impossible, because the classes clearly overlap.

However a few annotations on these statistics should be made. From class C patient number 73 has been left out, because his value for μ_A — which is 168.3 — deviated far too much from the rest of his class. The same applies to patient number 72 from class E, who has a μ_A of 336.5. Furthermore the number of patients differs a lot between classes: class A contains 129 patients, class C just 14, class D has 13 members and classes B and E each 10. For the use with neural networks and more precise statistics on the performance of the different methods for classification, larger and more equal numbers are required. Section 3.4 deals with this problem.

In figures 3.1 to 3.5 histograms for the selected parameters of class A are shown. Some of these histograms clearly display some kind of Gaussian function, while with a little imagination the other histograms could be seen as Gaussians as well. We therefore assume that the data of class A is distributed normally for all parameters, a property that is very useful when dealing with statistics. In statistical terms a data set with this property is said to be *multivariate normal*.

For the other classes the number of patients is much too small to draw the same conclusion, but we assume these classes are multivariate normal as well. In the remainder of this thesis the multivariate normal property will often be used.

3.4 Generation of data

In section 3.3 it can be found that the size of class A is much larger than the sizes of the other classes. Depending on the goal you want to achieve, this can be very awkward when using neural networks. If you're goal is to get a correct classification for as many patients as possible, independent of the class they belong to, there is no problem. If however you want to get a good classification ratio for each individual class, these differences in number of patients cause a problem, because the network focusses so much on the large class that the other classes are more or less neglected. Of course it is possible to reduce class A to

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	17.479	5.984	1.000	0.613	0.122	-0.123	0.461
β	0.442	0.066	0.613	1.000	0.303	-0.063	0.241
snr	2.175	0.032	0.122	0.303	1.000	-0.470	-0.144
S	4.786	0.726	-0.123	-0.063	-0.470	1.000	0.129
α_1	0.427	0.104	0.461	0.241	-0.144	0.129	1.000

Table 3.1: Estimated mean, standard deviation and correlation for class A.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	25.803	11.564	1.000	0.803	0.562	-0.606	0.178
β	0.399	0.070	0.803	1.000	0.614	-0.407	-0.038
snr	2.180	0.050	0.562	0.614	1.000	-0.219	-0.414
S	3.611	1.262	-0.606	-0.407	-0.219	1.000	-0.610
α_1	0.319	0.153	0.178	-0.038	-0.414	-0.610	1.000

Table 3.2: Estimated mean, standard deviation and correlation for class B.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	29.283	13.204	1.000	0.609	-0.020	-0.255	0.543
β	0.474	0.088	0.609	1.000	0.053	-0.167	0.635
snr	2.200	0.047	-0.020	0.053	1.000	-0.588	0.192
S	4.164	1.017	-0.255	-0.167	-0.588	1.000	-0.547
α_1	0.472	0.170	0.543	0.635	0.192	-0.547	1.000

Table 3.3: Estimated mean, standard deviation and correlation for class C.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	26.222	10.909	1.000	0.087	0.582	-0.364	0.098
β	0.425	0.102	0.087	1.000	0.132	-0.171	0.644
snr	2.197	0.044	0.582	0.132	1.000	-0.408	-0.006
S	4.105	1.104	-0.364	-0.171	-0.408	1.000	-0.523
α_1	0.466	0.207	0.098	0.644	-0.006	-0.523	1.000

Table 3.4: Estimated mean, standard deviation and correlation for class D.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	25.580	12.375	1.000	0.532	0.015	-0.466	0.468
β	0.453	0.100	0.532	1.000	0.200	-0.155	0.271
snr	2.218	0.031	0.015	0.200	1.000	-0.622	-0.360
S	4.063	1.155	-0.466	-0.155	-0.622	1.000	0.153
α_1	0.450	0.108	0.468	0.271	-0.360	0.153	1.000

Table 3.5: Estimated mean, standard deviation and correlation for class E.

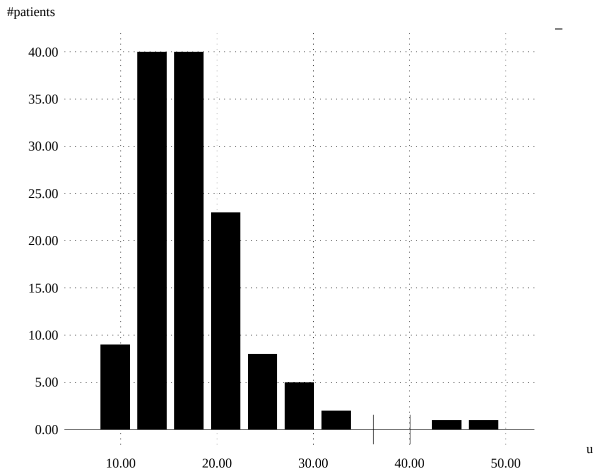


Figure 3.1: A's original distribution of μ.

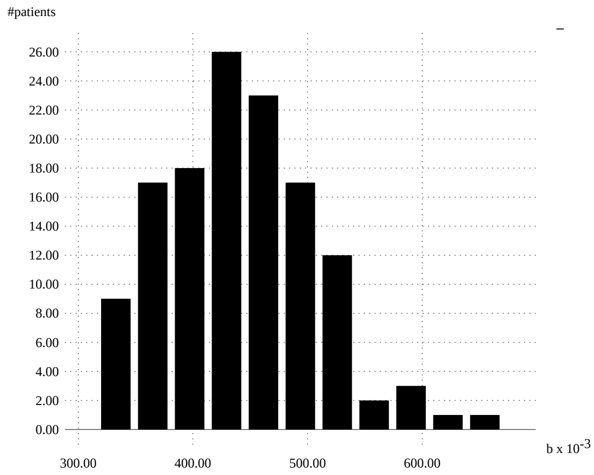


Figure 3.2: A's original distribution of β.

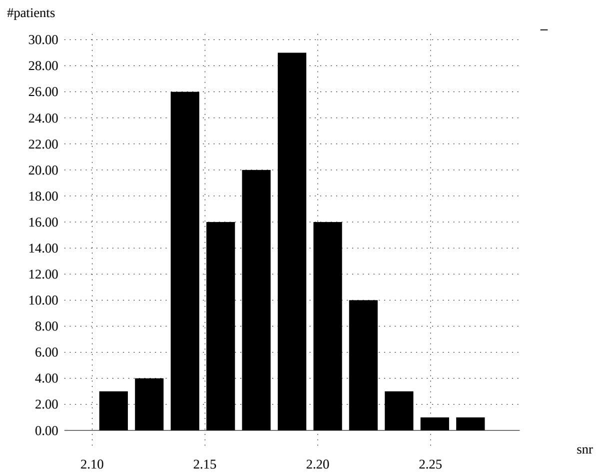


Figure 3.3: A's original distribution of snr.

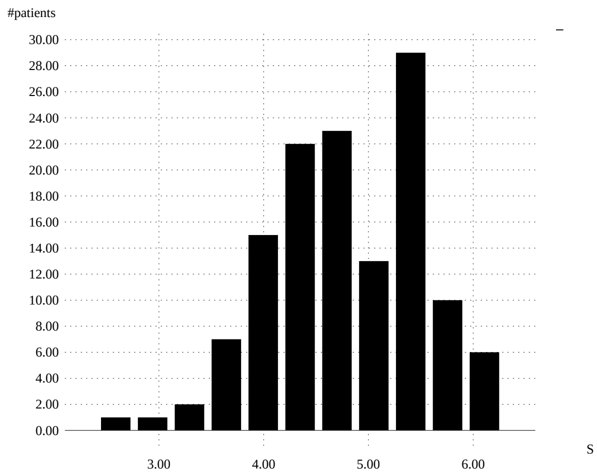


Figure 3.4: A's original distribution of S.

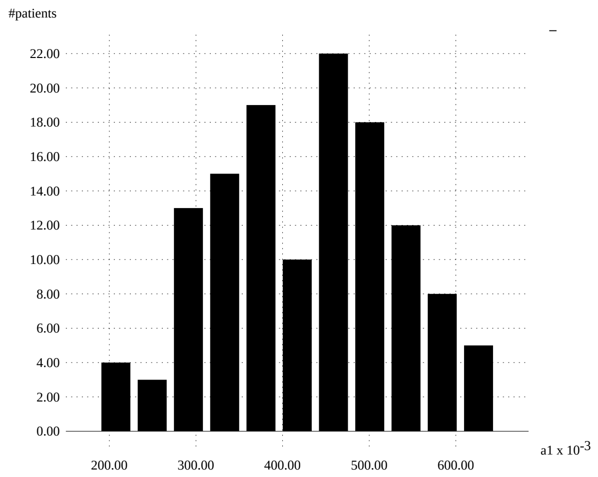


Figure 3.5: A's original distribution of α_1.

a size equal to the other sizes, but this way a lot of the diversity of class A will be lost. Furthermore it is very difficult to extract a set that is representative.

Another problem with the present-day classes is that they are small. The possibilities for measuring the performance of a classification method are limited, because the wrong classification of one patient can result in a decrease of performance of 10%.

The only solution to these problems seems to be the generation of additional data. For the smaller classes more data can be generated than for the larger classes, resulting in large classes of equal size. Data can be generated in a lot of ways. In section 3.4.1 a method will be examined that only uses statistical information about the original set, like the mean, the standard deviation and some correlation coefficients of the parameters. The method in section 3.4.2 uses both statistics and the original data.

3.4.1 A statistical method

This method only works if the data set that is to be enlarged is multivariate normal, an assumption that has been made in section 3.3. The statistics that are needed are the mean and standard deviation for each parameter of the class under observation. Furthermore, since it is possible with this method to correlate each parameter with one other parameter, some correlation coefficients are necessary as well. We have chosen to preserve the correlation between the most important parameter — μ_A — and all other parameters. All the necessary statistics can be found in the tables 3.1 to 3.5.

First a value for μ_A must be generated. Because the data is assumed to be normally distributed, a Gaussian random generator is needed. The steps necessary to simulate such a generator are the following.

1. Draw a x_n from a uniform distribution over the interval [0,1]. This can be done using an ordinary random generator.

2. Compute a Rayleigh distributed random variable y_n — see section 2.5.1 — using

$$y_n = \sqrt{2\sigma^2 \ln\left(\frac{1}{x_n}\right)} \tag{3.3}$$

with σ the standard deviation of the parameter to be generated.

3. Generate a pair of Gaussian random numbers w_n and w_{n+1} using

$$\begin{aligned} w_n &= y_n \cos(2\pi x_{n+1}) \\ w_{n+1} &= y_n \sin(2\pi x_{n+1}) \end{aligned} \tag{3.4}$$

Both these numbers should be used to get the desired Gaussian distribution.

To avoid extreme values, generated numbers with an absolute value greater than δ times σ are discarded. The value for δ can be chosen freely; high values result in data of great diversity, low values in data much more similar. Finally, to get the new sample for parameter μ_A, the mean of μ_A is added to the previously generated value w_n or w_{n+1}.

Now all the other parameters can be generated. To keep the original correlation between μ_A and the other parameters, the following construction is used. To simplify the explanation p represents the parameter to be generated, v refers to the actually generated value and ρ_{xy} stands for the correlation between parameter x and parameter y. A shift of the mean value of parameter p is achieved by

$$\mu_p' = \mu_p + \frac{v_{\mu_A} - \mu_{\mu_A}}{\sigma_{\mu_A}} \rho_{p\mu_A} \sigma_p \tag{3.5}$$

The standard deviation is adapted using

$$\sigma_p' = \left(1 - \left|\frac{v_{\mu_A} - \mu_{\mu_A}}{\sigma_{\mu_A}} \cdot \frac{\rho_{p\mu_A}}{\delta}\right|\right) \sigma_p \tag{3.6}$$

Since v_{μ_A} never deviates more from μ_{μ_A} than δ times σ_{μ_A}, σ_p' will always be significant. Notice that the new values for the mean (μ_p') and standard deviation (σ_p') of a parameter depend on v_{μ_A} and therefore differ per generated case.

The actual generation of the value for a parameter other than μ_A is now straightforward: the same procedure must be followed that was presented earlier for generating a value for μ_A, using the changed mean and standard deviation.

Statistics

Using the method described in section 3.4.1 a data set consisting of 1000 cases per class was generated, with δ set at a value of 2.0. Since 95% of the data set should be within at most 2σ distance of the mean per parameter, this would mean that 77% of the original data set with five parameters is covered. A global description of the program used — *statgen* — can be found in appendix A.

The statistics of the new sets can be found in tables 3.6 to 3.10. Comparison of these results with the statistics of the original sets shows that the new mean never deviates more from the old mean than 2%, positively or negatively. The new standard deviation is on the average 10–15% lower than the old standard deviation. This is probably caused by the 2σ limit that determines which patients are not included in the new set. This is an indication that for δ a value higher than 2.0 should be chosen.

The correlations between μ_A and the other parameters are fairly similar to the old correlation coefficients. The deviation lies between 0 and 10% for correlations with an absolute value over 0.1; for the smaller correlations the deviation percentage is high, but the absolute deviation is never more than 0.03. The correlation between the other parameters has no similarity to the old values anymore: not only are some relatively strong correlations lost, some parameters that were not correlated before now display a relatively strong correlation.

The histograms in figures 3.6 to 3.10, displaying the distribution of the five parameters of class A, show that the data is distributed more normally than before. This may seem to be a flaw of the method, but is simply the result of a much larger number of cases.

3.4.2 A kernel based method

Another method for the generation of data is described in [16, 17]. It uses a probability density K, the *kernel*, and a smoothing parameter h to approximate the distribution function of the data f by its n-point kernel estimate $f_{n,h,K}$. The symbol n represents the number of samples in the original data set on which the estimate will be based. The estimate is defined by

$$f_{n,h,K}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h} K\left(\frac{x - x_i}{h}\right) \tag{3.7}$$

with x_i representing the i-th sample from the original data set.

The characteristics of the generated data are controlled by the choice of K and h. For a normal density f with variance σ^2, which is the case in our problem, the optimal choice [16] for K turns out to be the Bartlett kernel (see figure 3.11), defined by

$$K(x) = \frac{3}{4}\left(1 - x^2\right) \tag{3.8}$$

in combination with a value for the smoothing parameter h given by

$$h = 1.8218\sigma n^{-\frac{1}{5}} \tag{3.9}$$

First the standard deviation and smoothing parameter for a parameter of the class that is to be enlarged must be determined. The standard deviation can be found in tables 3.1 to 3.5, while the smoothing parameter can be calculated using equation 3.9.

Secondly a sample is drawn from the density K, in this thesis the Bartlett distribution. For this purpose a Bartlett random generator must be constructed. The principle of this random generator is analogous to that of the Gaussian random generator: a variable y drawn from a uniform distribution on the interval

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	17.671	5.434	1.000	0.613	0.118	-0.102	0.511
β	0.443	0.058	0.613	1.000	0.108	-0.035	0.287
snr	2.175	0.028	0.118	0.108	1.000	-0.078	-0.016
S	4.782	0.621	-0.102	-0.035	-0.078	1.000	-0.027
α_1	0.428	0.091	0.511	0.287	-0.016	-0.027	1.000

Table 3.6: New mean, standard deviation and correlation for class A using the statistical method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	26.115	10.286	1.000	0.731	0.591	-0.620	0.179
β	0.401	0.067	0.731	1.000	0.444	-0.457	0.131
snr	2.181	0.043	0.591	0.444	1.000	-0.384	0.088
S	3.555	1.261	-0.620	-0.457	-0.384	1.000	-0.103
α_1	0.318	0.131	0.179	0.131	0.088	-0.103	1.000

Table 3.7: New mean, standard deviation and correlation for class B using the statistical method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	29.635	11.752	1.000	0.595	-0.029	-0.289	0.517
β	0.475	0.076	0.595	1.000	-0.033	-0.146	0.313
snr	2.200	0.041	-0.029	-0.033	1.000	-0.032	-0.037
S	4.138	0.858	-0.289	-0.146	-0.032	1.000	-0.181
α_1	0.473	0.146	0.517	0.313	-0.037	-0.181	1.000

Table 3.8: New mean, standard deviation and correlation for class C using the statistical method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	26.526	9.713	1.000	0.060	0.589	-0.390	0.109
β	0.422	0.084	0.060	1.000	0.063	-0.057	0.052
snr	2.198	0.039	0.589	0.063	1.000	-0.249	0.102
S	4.085	0.917	-0.390	-0.057	-0.249	1.000	-0.060
α_1	0.462	0.172	0.109	0.052	0.102	-0.060	1.000

Table 3.9: New mean, standard deviation and correlation for class D using the statistical method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	25.914	11.007	1.000	0.534	0.045	-0.502	0.463
β	0.454	0.085	0.534	1.000	0.034	-0.212	0.259
snr	2.218	0.027	0.045	0.034	1.000	-0.061	0.007
S	4.018	0.988	-0.502	-0.212	-0.061	1.000	-0.225
α_1	0.450	0.093	0.463	0.259	0.007	-0.225	1.000

Table 3.10: New mean, standard deviation and correlation for class E using the statistical method.

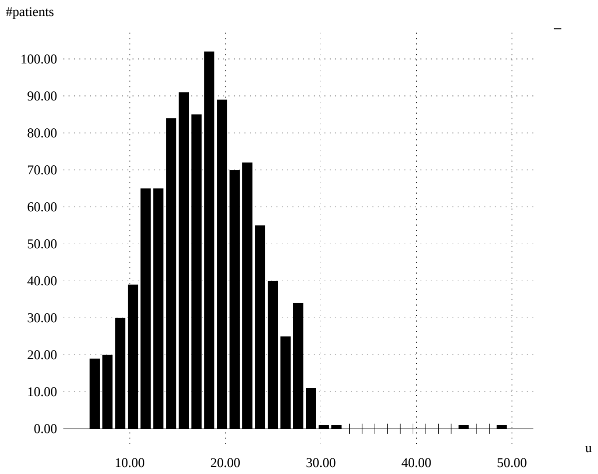


Figure 3.6: A's distribution of μ using the statistical method.

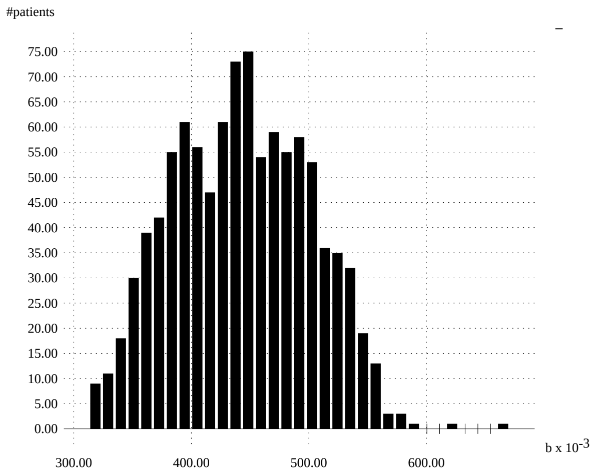


Figure 3.7: A's distribution of β using the statistical method.

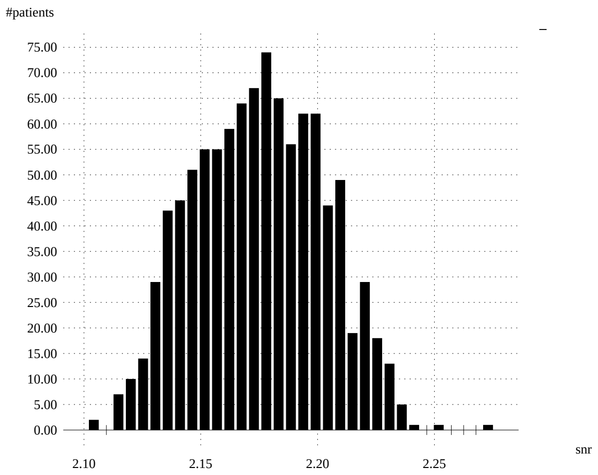


Figure 3.8: A's distribution of snr using the statistical method.

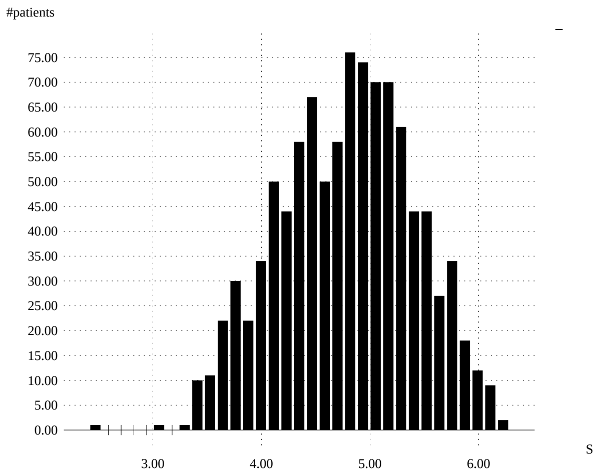


Figure 3.9: A's distribution of S using the statistical method.

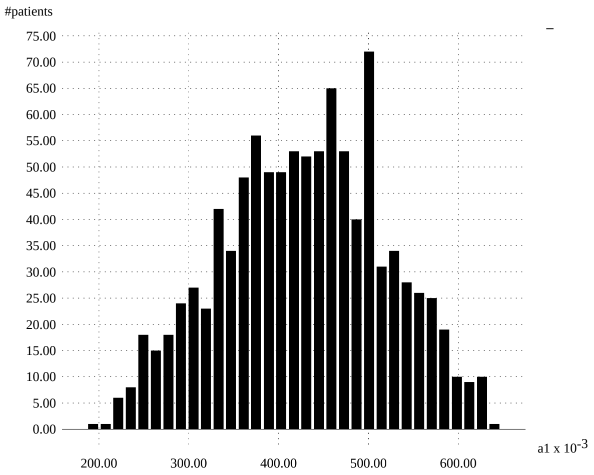


Figure 3.10: A's distribution of α_1 using the statistical method.

[0,1] is, using a certain function, transformed to a variable w that is distributed according to the Bartlett distribution.

The function that is needed by the generator is the inverse ($F^{-1}(y)$) of the cumulative Bartlett distribution function $F(x)$, which is given by

$$\begin{aligned} F(x) &= \int_{-1}^{x} \frac{3}{4}\left(1 - z^2\right) dz = \frac{3}{4}\left(z - \frac{1}{3}z^3\right)\Big|_{-1}^{x} \\ &= -\frac{1}{4}x^3 + \frac{3}{4}x + \frac{1}{2} \end{aligned} \tag{3.10}$$

Using Cardano's formula Maple[1] gives the following three solutions for $F^{-1}(y)$:

1. $X_2 + X_1$
2. $-\frac{1}{2}(X_2 + X_1) + \frac{\sqrt{3}}{2}(X_2 - X_1)i$
3. $-\frac{1}{2}(X_2 + X_1) - \frac{\sqrt{3}}{2}(X_2 - X_1)i$

where $X_1 = \sqrt[3]{1 - 2y - 2\sqrt{y^2 - y}}$ and $X_2 = \sqrt[3]{1 - 2y + 2\sqrt{y^2 - y}}$. Clearly the second and third solution are imaginary and thus useless, but the first might satisfy. Unfortunately the term $y^2 - y$ is not positive for a y from $[0, 1]$, which is the domain of interest, so the square root of this term is imaginary. This disqualifies the first solution also.

The only possible line of action to compute $F^{-1}(y)$ is that of numerical approximation. In this thesis Newton's method [18] is used because of its fast convergence. Since the method of Newton can only be used to determine the points where a function is zero, a function $g(x)$ is constructed that is zero for

[1]Maple is a computer algebra system developed at the University of Waterloo, Waterloo (Ontario), USA.

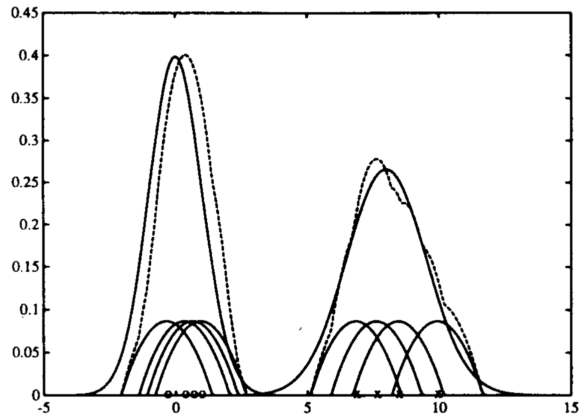


Figure 3.11: Approximation of a distribution function (solid line) displaying two classes (o's and x's). The kernel estimate (dashed line) is based on the Bartlett kernel.

x equal to $F^{-1}(y)$:

$$g(x) = F(x) - y = -\frac{1}{4}x^3 + \frac{3}{4}x + \frac{1}{2} - y \tag{3.11}$$

This function satisfies the conditions needed for the successful application of Newton's method, which are that on the interval [-1,1] $g'(x) \neq 0$ and $g''(x)$ is continuous[2], with $g'(x)$ representing the first-order and $g''(x)$ the second-order derivative of $g(x)$.

From a first value x_0, which we choose to be 0.5, better approximations of $w = F^{-1}(y)$ are computed using

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)} \tag{3.12}$$

With this method a precision of 0.005 is achieved with only four or five iterations.

Finally, an original data point x_i from the selected class is chosen with equal probability for each point, which will serve as a base for the new data point. The new data point v is now constructed by $v = x_i + hw$.

If the data one is dealing with is multi-dimensional — which is the case in this thesis — a few changes in the above procedure must be made. Both the smoothing parameter as well as the sample drawn from the kernel now become a vector with an entry for every parameter. The selected data point from the original set already is a vector, so the construction method described in the previous paragraph still applies.

[2] In fact holds $f'(-1) = f'(1) = 0$, but since the approximation is not started at one of the ends of the interval, this will not be a problem.

Statistics

To compare this method with the method described in section 3.4.1 a data set consisting of 1000 cases per class has been generated. In appendix A the program that was used — *kernelgen* — is briefly discussed.

Some statistics of the data sets generated with this method can be found in tables 3.11 to 3.15. Comparing these results with the original results — tables 3.1 to 3.5 — one can see that the mean of the enlarged data sets never deviates more from the original estimated mean than 2%. The new standard deviation is usually 5–10% higher than that of the original sets. A cause for this could be a too large value of the smoothing parameter.

The correlation between the parameters is completely lost for all classes. While in the original sets correlation coefficients of upto 0.8 occurred, in the new sets the correlation always stays within the interval [-0.1,0.1]. This is a second indication that the smoothing parameter should be decreased, for a smaller kernel results in data that is much more similar to the original data. However, this could cause the new set to be less multivariate normal, something that should be avoided.

Like in section 3.3 and 3.4.1 the distribution of every parameter is examined for class A. The histograms, figures 3.12 to 3.16, show that the way in which the data is distributed is a mixture between the original distribution and a pure normal distribution. This was to be expected, since not only the statistics but also the original data points were taken as basis for the new data.

The — important — decision which method to take for the generation of new data is not a straightforward one. The statistical method has the advantage that some correlation coefficients are maintained, but has no theoretical foundation. The kernel based method on the other hand has a theoretical background, but leads to complete loss of the correlation between the parameters. Tests using discriminant analysis as well as neural networks have not shown significant differences in classification results. Considering all these facts we have chosen to use the kernel based method because of its theoretical basis and the fact that it has been described in the literature.

3.5 Further formatting

The format of data is not suitable yet to feed to a neural network. As mentioned before, a big difference in the number of patients per class could cause the network to focus itself on the correct classification of the largest class, while neglecting the smaller classes; it was for this reason that additional data has been generated. For the same reason however, a big difference in the variance between the parameters is undesirable.

Suppose one parameter varies between 0 and 1, while another parameter varies between 0 and 1000. The error function a neural network tries to minimize decreases much more when the second parameter is optimized than when the first parameter is optimized, although the value of the first parameter could be much more significant for the classification process. The only way to solve this problem is through scaling.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	17.589	5.927	1.000	0.065	0.034	-0.044	0.083
β	0.442	0.066	0.065	1.000	0.027	0.007	-0.017
snr	2.175	0.032	0.034	0.027	1.000	-0.020	0.034
S	4.778	0.778	-0.044	0.007	-0.020	1.000	-0.023
α_1	0.430	0.108	0.083	-0.017	0.034	-0.023	1.000

Table 3.11: New mean, standard deviation and correlation for class A using the kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	25.462	12.365	1.000	-0.002	0.050	0.027	0.041
β	0.396	0.073	-0.002	1.000	-0.046	0.029	0.028
snr	2.182	0.054	0.050	-0.046	1.000	0.016	0.059
S	3.625	1.339	0.027	0.029	0.016	1.000	0.035
α_1	0.324	0.168	0.041	0.028	0.059	0.035	1.000

Table 3.12: New mean, standard deviation and correlation for class B using the kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	30.267	14.185	1.000	0.062	-0.003	-0.017	-0.037
β	0.477	0.092	0.062	1.000	-0.003	0.035	-0.106
snr	2.201	0.051	-0.003	-0.003	1.000	-0.025	0.001
S	4.118	1.124	-0.017	0.035	-0.025	1.000	0.031
α_1	0.471	0.183	-0.037	-0.106	0.001	0.031	1.000

Table 3.13: New mean, standard deviation and correlation for class C using the kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	26.223	11.617	1.000	-0.070	0.039	-0.016	-0.053
β	0.422	0.109	-0.070	1.000	0.011	0.013	0.023
snr	2.197	0.046	0.039	0.011	1.000	0.018	0.062
S	4.109	1.193	-0.016	0.013	0.018	1.000	0.018
α_1	0.465	0.221	-0.053	0.023	0.062	0.018	1.000

Table 3.14: New mean, standard deviation and correlation for class D using the kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	25.491	12.875	1.000	0.052	-0.053	0.015	0.011
β	0.447	0.103	0.052	1.000	-0.050	0.052	-0.029
snr	2.217	0.034	-0.053	-0.050	1.000	0.010	-0.008
S	4.077	1.269	0.015	0.052	0.010	1.000	0.032
α_1	0.450	0.116	0.011	-0.029	-0.008	0.032	1.000

Table 3.15: New mean, standard deviation and correlation for class E using the kernel based method.

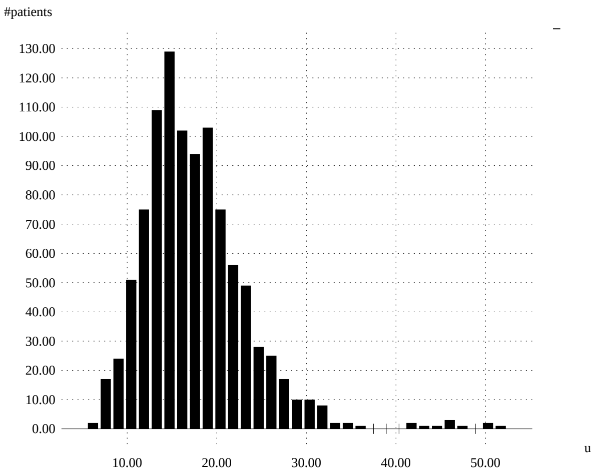


Figure 3.12: A's distribution of μ using the kernel based method.

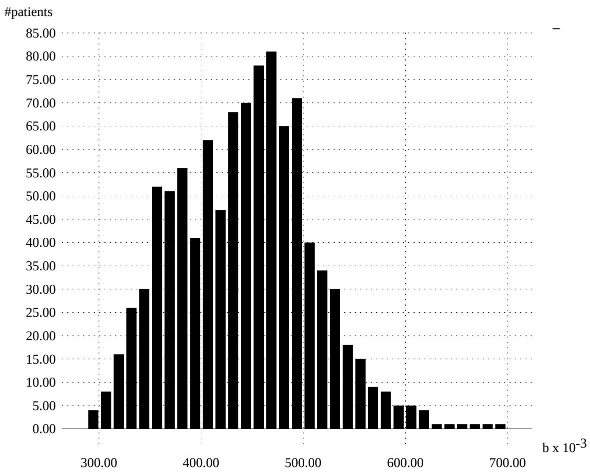


Figure 3.13: A's distribution of β using the kernel based method.

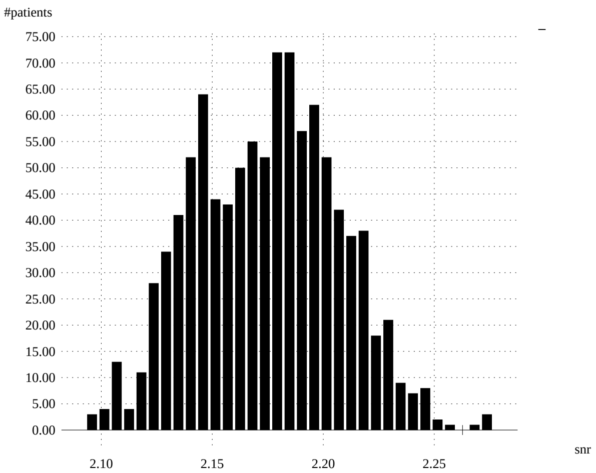


Figure 3.14: A's distribution of *snr* using the kernel based method.

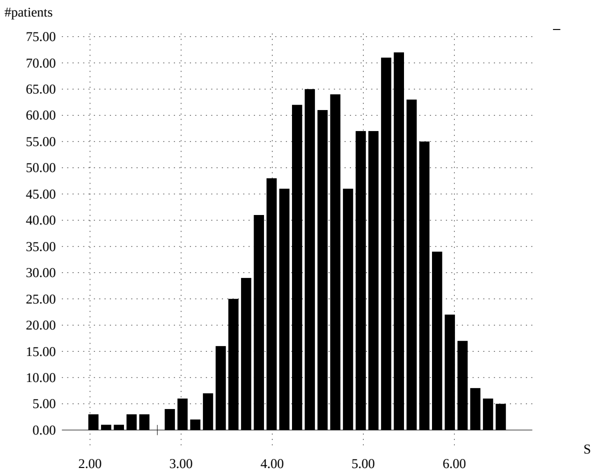


Figure 3.15: A's distribution of S using the kernel based method.

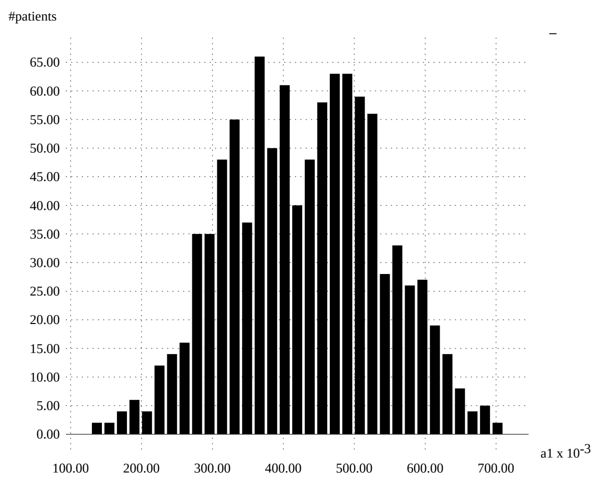


Figure 3.16: A's distribution of α_1 using the kernel based method.

There seem to be two sensible scaling techniques. The first projects parameterwise the minimum (maximum) value of the data set to the minimum (maximum) of the new domain. A disadvantage of this method is that a single extreme value in the data set could cause the other data points to be highly concentrated in only a small part of the new domain. The second technique therefore projects the mean minus (plus) for instance 2 times the standard deviation to the minimum (maximum) of the new domain. The data points that deviate more from the mean value are projected to the ends of the new domain as well. This way some information is lost, but the other data points are spread more equal over the domain.

In this thesis the second technique is used. From all the data to be classified — this operation is not executed on one class at the time, like generation for example — μ and σ are computed. Then $\mu - 2\sigma$ is projected on 0.1 and $\mu + 2\sigma$ on 0.9. The 2σ boundary is chosen because this way no more than 5% of the information should be lost. The new domain [0.1,0.9] is chosen because these values are useful when dealing with back-propagation networks (see section 5.4.2). A description of the program performing this operation — *format* — can be found in appendix A.

After scaling the data is divided in two sets of equal size. Since 1000 samples are available for each class, both sets contain 500 cases per class. One set, the training set, contains only generated cases. The training set is used to train the network or, in the case of discriminant analysis, for calibration purposes. In the other set, the test set, all real — i.e. not generated — data is contained, completed with some generated cases. This set is used to trace the behavior of the network when several parameters of it are varied. A subset of the test set, the validation set, contains only the real patients. This set is used to measure the actual performance of the method under observation for comparison with the other methods.

Chapter 4

Discriminant analysis

This chapter is about discriminant analysis, a statistical method enabling a patient to be classified into one of a set of diseases on the basis of a laboratory profile. In the first part of this chapter the method is described without discussing the complete theory behind it, for more details see [19]. In the second part the results obtained with discriminant analysis are given.

4.1 Principles of the method

Discriminant analysis can be performed using two methods, a *parametric* and a *non-parametric* method. If the data to be dealt with is multivariate normal the parametric method is used, otherwise the non-parametric method is applied. Since we have assumed that our data is multivariate normal, only the parametric method is explained.

Discriminant analysis divides a multi-dimensional observation-space into several regions, one for each disease to be recognized. The division is made using *profiles*, i.e. data that has already been classified. After the borders between the different classes have been determined, new data can be classified by observing in what partition the data is situated.

The partitioning of the observation-space can be achieved in many ways, some of which are better than others. If the data does not overlap, the borders between the different classes are straightforward, but this is usually not the case. The trick is to divide the space in such a way that as many patients of the profile as possible are classified correctly — see figure 4.1. If the data in the profile is representative for the classes to be discriminated, new patients will now be classified as good as possible also.

Classification rules on which the discrimination is carried out are derived from several probabilities. First of all the prior probability ($p(D_i)$) for each class is necessary. With the prior probability the belief that a patient belongs to a certain class, before knowing the result of a laboratory test, is expressed. Usually two different values for the prior probability are considered, depending on the goal you want to achieve. If you want to classify correctly as many patients as possible, without regarding the classification results per class, the prior probability of a class is set at the fraction of the profiles that are patients

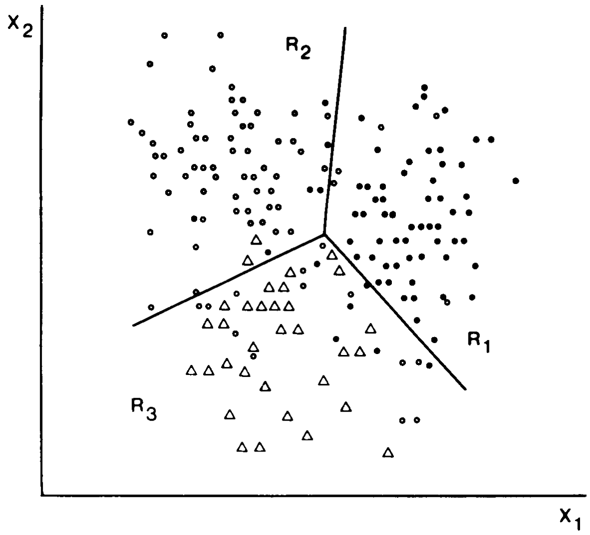


Figure 4.1: Partitioning of a 2-dimensional observation space into three regions.

belonging to that class. If however you want to get a high percentage of correctly classified patients per class, all prior probabilities are chosen equal, adding up to one.

Another probability that is necessary is the group-specific density, $p(x|D_i)$, which defines the probability that a parameter-vector x actually occurs for a patient belonging to a disease group D_i. For multivariate normal data this probability density can be described by

$$p(x|D_i) = (2\pi)^{-\frac{k}{2}}|V_i|^{-\frac{1}{2}}e^{-\frac{1}{2}(x-\mu_i)^T V_i^{-1}(x-\mu_i)} \tag{4.1}$$

where V_i represents the covariance matrix of class i, $|V_i|$ is the determinant of it, k equals the dimension of vector x and x^T is the transposed version of x.

Using Bayes' theorem, the posterior probability can be calculated using

$$p(D_i|x) = \frac{p(D_i)p(x|D_i)}{\sum\limits_g (p(D_g)p(x|D_g))} \tag{4.2}$$

with g an index over all classes. With the posterior probability the probability is meant that a patient, represented by his parameter-vector x, suffers from disease D_i. The posterior probability, which can be calculated for an arbitrary patient, determines to which class a patient is assigned: the patient under observation is classified into the class with the highest posterior probability.

Some software packets [20, p. 677-771] that can perform discriminant analysis use a slightly different, but related method. They try to minimize the *generalized squared distance* $d_i^2(x)$, defined by

$$d_i^2(x) = (x-\mu_i)^T V_i^{-1}(x-\mu_i) - 2\log_e p(D_i) + \log_e |V_i| \tag{4.3}$$

It can easily be checked using equations 4.1 and 4.2 that the following relation holds:

$$p(D_i|x) = \frac{e^{-\frac{1}{2}d_i^2(x)}}{\sum\limits_g e^{-\frac{1}{2}d_g^2(x)}} \tag{4.4}$$

It is obvious that a minimal $d_i^2(x)$ conforms to a maximal $p(D_i|x)$.

An advantage of using generalized squared distances is that, depending on certain choices, certain terms can be left out. If for instance all prior probabilities are equal — either because the number of patients is equal for all classes, or because they are chosen equal to get a good classification percentage for every class — the term $-2\log_e p(D_i)$ can be omitted, since it has the same value for each class.

Another term that can sometimes be left out is $\log_e |V_i|$. If the variances and covariances of the parameters are independent of the disease, the above term is equal for every class and can thus be omitted. This condition can be imposed on the discriminant analysis by stating that instead of the within-group covariance matrices V_i the pooled covariance matrix V must be used. With the pooled covariance matrix the variances and covariances of all the profiles together are meant. Discriminant analysis based on this principle is called *linear discriminant analysis*. If however the covariance matrix is dependent on the type of disease, this reduction is not allowed. This type of discriminant analysis is often called *quadratic discriminant analysis*.

4.2 Results

The results that are presented in the remainder of this chapter have been obtained using a statistical analysis system called SAS/STAT[1]. More information about the procedure used (*DISCRIM*) can be found in [20].

4.2.1 Measuring the performance

The performance of discriminant analysis is measured using so called *confusion* or *specificity/sensitivity* matrices, displaying the classification percentages of the data set under observation. At position (i, j), where i represents the i-th row and j the j-th column, the percentage of patients *belonging* to class i that are *classified into* class j can be found. Since each patient of a certain class is classified into some class, the percentages in every row add up to 100%. On the diagonal the percentage of correctly classified patients can be found, so if a method is said to classify well, these percentages should be high.

In medicine often a group of healthy people is compared to one group of people suffering from some disease. In the resulting matrix the percentage "true negatives" — i.e. healthy people that are classified as being healthy — is called the *specificity* of a parameter set, while the percentage "true positives" — ill people that are classified as being ill — is called the *sensitivity* of that parameter set. The goal that is pursued determines which percentage is to be optimized: if as little healthy people as possible should be classified as being ill in order

[1]SAS/STAT has been developed at SAS Institute Inc., Cary (NC), USA.

to avoid unnecessary examination, which could be damaging or very costly, the specificity should be high; if as little ill people as possible should be classified as being in good health in order to avoid erroneously aborting further treatment, the sensitivity should be high. Although always both goals are striven for, usually they can not be achieved at the same time. The reason for this is that if the specificity should be optimized, in doubtful cases a patient is classified as being healthy, resulting in a better specificity but worse sensitivity, while for optimizing the sensitivity the opposite decision should have been made.

4.2.2 Approach to solving the problem

In sections 3.1 and 3.2 our general problem, defined in section 1.3, was filled in by selecting five classes that had to be discriminated on the basis of five parameters. But before proceeding with this problem, first the ability to discriminate between only two classes is investigated. One of these classes is always class A containing healthy people, while for the other class successively class B to E — the four different diseases — is chosen. If these partial problems can not be solved adequately, there is little chance of solving the overall problem.

As will be explained in chapters 5 and 6, the way in which neural networks classify data is completely different from discriminant analysis as we have described so far. Neural networks use a training set on the basis of which the problem is learned, and a test set to check if the problem has been learned properly. Once a network has been found that seems to solve the problem reasonably well, a validation set is presented to it to determine the performance of that particular network. To get a fair comparison between neural networks and discriminant analysis a similar procedure should be followed, using the same data sets. Since discriminant analysis always first partitions the observation space in regions and then classifies a patient on base of the region he is situated in, the above procedure can be exactly imitated. Furthermore the same set with which the neural networks are trained — the training set, described at the end of section 3.5 — is used for calibrating the observation space. Hereafter both the test and the validation set — also see section 3.5 — are classified using the acquired partitioning.

In contrast to neural networks for discriminant analysis it is not necessary to have classes of approximately equal size if the goal is to get a high percentage of correct classification for each class. Therefore it is possible to work both with the original data as well as with the generated data; if the original data is used the classification results are more reliable because no errors are introduced by the generation process, but for a fair comparison with neural networks the generated data should be used. We have chosen to use both sets because this way it may be possible to draw some conclusions about how well the original data is represented by the generated data.

When using discriminant analysis for the classification of data, several decisions are available, some of which have already been described in section 4.1. The first option concerns the type of discriminant analysis: both linear discriminant analysis as well as the quadratic variant can used. Secondly, depending on the goal that is to be achieved, the prior probability of each class can be chosen proportional to the size of the class or equal to the other classes. When performing discriminant analysis using the generated data this option does not apply, since every training set contains for each class the same number of patients, always resulting in equal prior probabilities. If the data on the basis of which the observation space is partitioned and the data that is to be classified are the same[2] — this is only the case when classifying the original data — the choice has to be made whether classification should take place using *resubstitution* or *cross-validation*. Resubstitution means that first the observation space is partitioned on the basis of all profiles, after which the same profiles are classified. The cross-validation or jackknife method partitions the observation space using all but one profiles, after which the remaining profile is classified; this process has to be repeated for every profile. It can be proven that the percentage of correctly classified patients using the cross-validation method is at most equal to the percentage obtained with the resubstitution method [19, p. 124].

The results on the first problems solved with discriminant analysis are used in two ways. Of course the results are needed for comparing discriminant analysis with neural networks, but they are also used to get a feeling for the different options in discriminant analysis. When dealing with the other problems, this experience will be used to get the best results without doing superfluous work.

4.2.3 Discrimination between A and B

In this section the results of the discrimination between classes A and B are presented. All the options of discriminant analysis that have been mentioned in section 4.2.2 are tried to get a complete view on the method. For a discussion of the results is referred to section 4.3.

Classifying the original data using proportional prior probabilities

The results of linear discriminant analysis can be found in confusion matrix 4.5 when resubstitution is applied, and in matrix 4.6 when cross-validation is applied. Since class A contains 129 patients and class B only 10, the prior probabilities are set at $\frac{129}{139}$ and $\frac{10}{139}$.

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 100.0 & 0.0 \\ & \\ 30.0 & 70.0 \end{pmatrix} \end{array} \quad (4.5) \qquad \begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 98.5 & 1.5 \\ & \\ 40.0 & 60.0 \end{pmatrix} \end{array} \quad (4.6)$$

When performing quadratic discriminant analysis, the results for resubstitution are presented in matrix 4.8, and for cross-validation in matrix 4.8.

[2] In this case discriminant analysis is said to be *retrospective*.

$$\begin{array}{c|cc} & A & B \\ \hline A & \multicolumn{2}{c}{\multirow{2}{*}{}} \end{array}$$

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 100.0 & 0.0 \\ & \\ 10.0 & 90.0 \end{pmatrix} \end{array} \tag{4.7}$$

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 100.0 & 0.0 \\ & \\ 50.0 & 50.0 \end{pmatrix} \end{array} \tag{4.8}$$

Classifying the original data using equal prior probabilities

If equal prior probabilities are chosen, the results of linear discriminant analysis using resubstitution and cross-validation can be found in matrix 4.9 respectively 4.10.

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 95.4 & 4.6 \\ & \\ 10.0 & 90.0 \end{pmatrix} \end{array} \tag{4.9}$$

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 94.6 & 5.4 \\ & \\ 30.0 & 70.0 \end{pmatrix} \end{array} \tag{4.10}$$

Quadratic discriminant analysis yields results for resubstitution as in matrix 4.11, and for cross-validation as in matrix 4.12.

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 96.9 & 3.1 \\ & \\ 10.0 & 90.0 \end{pmatrix} \end{array} \tag{4.11}$$

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 96.9 & 3.1 \\ & \\ 30.0 & 70.0 \end{pmatrix} \end{array} \tag{4.12}$$

Classifying the generated data

The results of applying linear discriminant analysis to the generated data can be found in matrices 4.13 to 4.15 for respectively the training, test and validation set. The confusion matrix of the training or calibration set has been acquired using resubstitution.

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 86.2 & 13.8 \\ & \\ 23.0 & 77.0 \end{pmatrix} \end{array} \tag{4.13}$$

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 88.4 & 11.6 \\ & \\ 25.2 & 74.8 \end{pmatrix} \end{array} \tag{4.14}$$

$$\begin{array}{cc} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 89.2 & 10.8 \\ & \\ 10.0 & 90.0 \end{pmatrix} \end{array} \tag{4.15}$$

If not linear but quadratic discriminant analysis is applied to the same data sets, the results on the calibration, test and validation set are as presented in the confusion matrices 4.16 to 4.18.

$$\begin{array}{c} \\ A \\ \\ B \end{array} \begin{array}{c} \begin{array}{cc} A & B \end{array} \\ \begin{pmatrix} 88.6 & 11.4 \\ & \\ 16.0 & 84.0 \end{pmatrix} \end{array} \quad (4.16)$$

$$\begin{array}{c} \\ A \\ \\ B \end{array} \begin{array}{c} \begin{array}{cc} A & B \end{array} \\ \begin{pmatrix} 90.2 & 9.8 \\ & \\ 16.6 & 83.4 \end{pmatrix} \end{array} \quad (4.17)$$

$$\begin{array}{c} \\ A \\ \\ B \end{array} \begin{array}{c} \begin{array}{cc} A & B \end{array} \\ \begin{pmatrix} 89.8 & 10.1 \\ & \\ 0.0 & 100.0 \end{pmatrix} \end{array} \quad (4.18)$$

4.2.4 Discrimination between A and C

In this section the results of discriminant analysis on the problem of discerning between class A and C are presented. In contrast to the previous section only quadratic discriminant analysis is used because the results are clearly better than those obtained with linear discriminant analysis. A discussion of the results can be found in section 4.3.

Classifying the original data using proportional prior probabilities

With the prior probabilities are set proportional to the size of the classes, so for class A to $\frac{129}{143}$ and for class C to $\frac{14}{143}$, the results of — quadratic — discriminant analysis are presented in matrix 4.19 for the resubstitution method, and in matrix 4.20 for the cross-validation method.

$$\begin{array}{c} \\ A \\ \\ C \end{array} \begin{array}{c} \begin{array}{cc} A & C \end{array} \\ \begin{pmatrix} 96.9 & 3.1 \\ & \\ 50.0 & 50.0 \end{pmatrix} \end{array} \quad (4.19)$$

$$\begin{array}{c} \\ A \\ \\ C \end{array} \begin{array}{c} \begin{array}{cc} A & C \end{array} \\ \begin{pmatrix} 96.9 & 3.1 \\ & \\ 64.3 & 35.7 \end{pmatrix} \end{array} \quad (4.20)$$

Classifying the original data using equal prior probabilities

Matrix 4.21 displays the results of discriminant analysis for equal prior probabilities using resubstitution, while in matrix 4.22 the results using cross-validation can be found.

$$\begin{array}{c} \\ A \\ \\ C \end{array} \begin{array}{c} \begin{array}{cc} A & C \end{array} \\ \begin{pmatrix} 91.5 & 8.5 \\ & \\ 28.6 & 71.4 \end{pmatrix} \end{array} \quad (4.21)$$

$$\begin{array}{c} \\ A \\ \\ C \end{array} \begin{array}{c} \begin{array}{cc} A & C \end{array} \\ \begin{pmatrix} 87.6 & 12.4 \\ & \\ 57.1 & 42.9 \end{pmatrix} \end{array} \quad (4.22)$$

Classifying the generated data

Quadratic discriminant analysis applied to the generated data results in matrices 4.23 to 4.25, presenting the results of the calibration set, which have been acquired using resubstitution, the test set and the validation set.

$$\begin{array}{cc} & \begin{array}{cc} A & C \end{array} \\ \begin{array}{c} A \\ C \end{array} & \begin{pmatrix} 91.0 & 9.0 \\ 16.6 & 83.4 \end{pmatrix} \end{array} \quad (4.23) \qquad \begin{array}{cc} & \begin{array}{cc} A & C \end{array} \\ \begin{array}{c} A \\ C \end{array} & \begin{pmatrix} 89.6 & 10.4 \\ 18.4 & 81.6 \end{pmatrix} \end{array} \quad (4.24)$$

$$\begin{array}{cc} & \begin{array}{cc} A & C \end{array} \\ \begin{array}{c} A \\ C \end{array} & \begin{pmatrix} 80.6 & 19.4 \\ 35.7 & 64.3 \end{pmatrix} \end{array} \quad (4.25)$$

4.2.5 Discrimination between A and D

The results of discriminating between class A and D can be found in this section; the results are discussed in section 4.3.

Classifying the original data using proportional prior probabilities

Discriminant analysis where A's prior probability is set at $\frac{129}{142}$ and C's prior probability at $\frac{13}{142}$ yields the confusion matrices 4.26 and 4.27 for the resubstitution respectively the cross-validation method.

$$\begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ D \end{array} & \begin{pmatrix} 97.7 & 2.3 \\ 30.8 & 69.2 \end{pmatrix} \end{array} \quad (4.26) \qquad \begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ D \end{array} & \begin{pmatrix} 96.9 & 3.1 \\ 53.8 & 46.2 \end{pmatrix} \end{array} \quad (4.27)$$

Classifying the original data using equal prior probabilities

If equal instead of proportional prior probabilities are used, the performance of discriminant analysis using resubstitution can be found in matrix 4.28, while cross-validation produces matrix 4.29.

$$\begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ D \end{array} & \begin{pmatrix} 90.7 & 9.3 \\ 30.8 & 69.2 \end{pmatrix} \end{array} \quad (4.28) \qquad \begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ D \end{array} & \begin{pmatrix} 88.4 & 11.6 \\ 38.5 & 61.5 \end{pmatrix} \end{array} \quad (4.29)$$

Classifying the generated data

On the generated data discriminant analysis results in matrices 4.30 to 4.32, showing the performance on the calibration, test and validation set.

$$\begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ \\ D \end{array} & \begin{pmatrix} 88.8 & 11.2 \\ & \\ 24.6 & 75.4 \end{pmatrix} \end{array} \tag{4.30}$$

$$\begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ \\ D \end{array} & \begin{pmatrix} 88.2 & 11.8 \\ & \\ 19.8 & 80.2 \end{pmatrix} \end{array} \tag{4.31}$$

$$\begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ \\ D \end{array} & \begin{pmatrix} 83.0 & 17.0 \\ & \\ 30.8 & 69.2 \end{pmatrix} \end{array} \tag{4.32}$$

4.2.6 Discrimination between A and E

In this section the performance of discriminant analysis on the problem of discriminating between class A and E are given. The results are discussed in section 4.3.

Classifying the original data using proportional prior probabilities

If the prior probabilities for class A and E are set at $\frac{129}{139}$ and $\frac{10}{139}$, the performance of discriminant analysis using resubstitution can be found in matrix 4.33, while cross-validation yields matrix 4.34.

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 97.7 & 2.3 \\ & \\ 40.0 & 60.0 \end{pmatrix} \end{array} \tag{4.33}$$

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 96.9 & 3.1 \\ & \\ 80.0 & 20.0 \end{pmatrix} \end{array} \tag{4.34}$$

Classifying the original data using equal prior probabilities

The results of discriminant analysis with equal prior probabilities can be found in matrices 4.35 and 4.36, where the results have been obtained using resubstitution respectively cross-validation.

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 86.1 & 13.9 \\ & \\ 20.0 & 80.0 \end{pmatrix} \end{array} \tag{4.35}$$

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 85.3 & 14.7 \\ & \\ 70.0 & 30.0 \end{pmatrix} \end{array} \tag{4.36}$$

Classifying the generated data

In matrices 4.37 to 4.39 the results of discriminant analysis on the training, test and validation set are presented. Since the observation space has been partitioned on the basis of the calibration set, the performance on that calibration set has been acquired using resubstitution.

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 87.8 & 12.2 \\ & \\ 17.8 & 82.2 \end{pmatrix} \end{array} \qquad (4.37)$$

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 87.2 & 12.8 \\ & \\ 15.6 & 84.8 \end{pmatrix} \end{array} \qquad (4.38)$$

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 77.5 & 22.5 \\ & \\ 20.0 & 80.0 \end{pmatrix} \end{array} \qquad (4.39)$$

4.2.7 Discrimination between A, B, C, D and E

The problem of discriminating between all five classes simultaneously using discriminant analysis is dealt with in this section. However, in this section only the results are presented, while a discussion on the results can be found in section 4.3.

Classifying the original data using proportional prior probabilities

For the five-class problem having proportional prior probabilities means that the prior probabilities for class A to E successively are set at $\frac{129}{176}$, $\frac{10}{176}$, $\frac{14}{176}$, $\frac{13}{176}$ and $\frac{10}{176}$. Quadratic discriminant analysis in this case gives matrix 4.40 for the resubstitution method, and matrix 4.41 for the cross-validation method.

$$\begin{array}{cc} & \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{pmatrix} 96.1 & 0.0 & 3.1 & 0.0 & 0.8 \\ 10.0 & 90.0 & 0.0 & 0.0 & 0.0 \\ 42.9 & 7.1 & 42.9 & 7.1 & 0.0 \\ 30.8 & 0.0 & 15.4 & 38.4 & 15.4 \\ 40.0 & 10.0 & 20.0 & 10.0 & 20.0 \end{pmatrix} \end{array} \qquad (4.40)$$

$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{pmatrix}
95.4 & 0.0 & 3.1 & 0.0 & 1.5 \\
20.0 & 30.0 & 0.0 & 40.0 & 10.0 \\
42.9 & 7.1 & 0.0 & 28.6 & 21.4 \\
38.4 & 0.0 & 15.4 & 23.1 & 23.1 \\
50.0 & 20.0 & 20.0 & 10.0 & 0.0
\end{pmatrix}
\end{array}
\tag{4.41}
$$

Classifying the original data using equal prior probabilities

If all prior probabilities are chosen equal to each other, the performance of discriminant analysis can be found in matrix 4.42 if resubstitution is used, and in matrix 4.43 if cross-validation is used.

$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{pmatrix}
81.4 & 2.3 & 4.7 & 2.3 & 9.3 \\
10.0 & 90.0 & 0.0 & 0.0 & 0.0 \\
28.6 & 14.3 & 35.7 & 7.1 & 14.3 \\
23.1 & 0.0 & 23.1 & 38.5 & 15.4 \\
20.0 & 10.0 & 10.0 & 0.0 & 60.0
\end{pmatrix}
\end{array}
\tag{4.42}
$$

$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{pmatrix}
79.8 & 2.3 & 4.7 & 3.9 & 9.3 \\
10.0 & 40.0 & 0.0 & 40.0 & 10.0 \\
28.6 & 14.3 & 0.0 & 28.6 & 28.6 \\
23.1 & 0.0 & 30.8 & 23.1 & 23.1 \\
30.0 & 20.0 & 30.0 & 20.0 & 0.0
\end{pmatrix}
\end{array}
\tag{4.43}
$$

Classifying the generated data

On the calibration, test and validation set discriminant analysis yields the results presented in confusion matrices 4.44 to 4.46. Just like with the previous problems matrix 4.44, which shows the performance on the calibration set, has been acquired using resubstitution.

$$
\begin{array}{cc}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{pmatrix}
82.4 & 6.0 & 1.8 & 1.8 & 8.0 \\
13.4 & 56.2 & 5.6 & 6.4 & 18.4 \\
12.2 & 11.0 & 42.6 & 10.2 & 24.0 \\
21.0 & 21.8 & 19.2 & 21.2 & 16.8 \\
18.0 & 9.0 & 17.8 & 6.6 & 48.6
\end{pmatrix}
\end{array}
\tag{4.44}
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{pmatrix}
83.0 & 4.6 & 3.0 & 1.2 & 8.2 \\
13.4 & 56.0 & 7.2 & 7.4 & 16.0 \\
13.8 & 11.6 & 39.8 & 9.8 & 25.0 \\
15.8 & 25.2 & 20.4 & 17.2 & 21.4 \\
14.8 & 11.4 & 19.4 & 9.0 & 45.4
\end{pmatrix}
\end{array}
\tag{4.45}
$$

$$
\begin{array}{cc}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\begin{pmatrix}
64.4 & 2.3 & 15.5 & 2.3 & 15.5 \\
0.0 & 70.0 & 10.0 & 10.0 & 10.0 \\
28.6 & 14.3 & 35.7 & 7.1 & 14.3 \\
38.4 & 30.8 & 15.4 & 7.7 & 7.7 \\
30.0 & 0.0 & 30.0 & 10.0 & 30.0
\end{pmatrix}
\end{array}
\tag{4.46}
$$

4.3 Discussion

On the basis of the results presented in the sections 4.2.3 to 4.2.7 several observations can be made, some of which are obvious, while others are not.

First of all in section 4.2.3 quadratic discriminant analysis has shown to be superior to linear discriminant analysis, both when dealing with the original data set as well as with the generated data. This was to be expected since linear discriminant analysis assumes that for all classes the covariance matrices are approximately equal. As can be found in section 3.3, this assumption is not true.

Another observation that can be made is that the performance of discriminant analysis on the original data sets gets worse when cross-validation instead of resubstitution is used. This observation is in agreement with the theory which states that the percentage of correctly classified patients using the cross-validation method is at most equal to the percentage obtained with the resubstitution method [19, p. 124]. The deterioration of the percentage of correctly classified patients is much stronger for the small classes than for class A, something that can be explained very easily. For a class that consists of only a few patients which are scattered over a relatively large part of the observation space like class E, each patient has a big influence on the partitioning of the observation space. If now the observation space is partitioned without regarding such a patient as is the case for cross-validation, classification of this patient will be bound to be erroneous because there are no patients causing the class boundary to be moved towards this patient. For a large class like class A this problem applies less because the probability that a patient has other patients of the same class in its neighbourhood is much larger.

A third obvious observation is that the choice of the prior probabilities — proportional or equal — has a big influence on the percentages of correct classification. As said before, proportional prior probabilities result in the correct classification of as many patients as possible independent of the class they belong to, while equal prior probabilities result in an as high as possible percentage of correctly classified patients per class. Since the difference in size of the classes is big — 10 to 14 against 129 patients — the difference in performance between discriminant analysis using proportional and equal prior probabilities will also be big.

Looking at the performance on the generated data it strikes that for all problems the confusion matrices of the calibration and the test set are more or less identical, see for instance matrices 4.44 and 4.45. This is not very surprising since both sets have been generated on the basis of the same data using the same procedure. The differences between the two sets have been caused by the random generator, but since both sets are large — 500 patients each — their distribution functions approximate the distribution function of the random generator. The similarity between the calibration or training set and the test set is very useful to detect over-learning, an aspect of neural networks that is explained in section 5.3.3.

As a result of the previous conclusion classification of the test set yields better results than classification of the validation set; since the observation space is partitioned in a way that is optimal for the calibration set, to which the test set is approximately equal, a nearly optimal classification is to be expected for the test set as well. The validation set on the other hand deviates much more from the calibration set, so on the average for this set the partitioning will not be optimal.

In order to draw some conclusions about the quality of the generating process the results of discriminant analysis on the original data set and on the validation set using a calibration set can be compared. Since in the calibration set all classes are represented by equally large numbers, it seems to be fair to choose equal prior probabilities for the comparison with the original data. If the results on the validation set are compared with the results on the original data using cross-validation, it strikes that the performance on the validation set is much higher. As mentioned before this is probably caused by the small number of patients for most classes in the original set, resulting in a bad performance on the original data set. Compared to the performance on the original set using resubstitution the performance on the validation set is somewhat lower, but more important is that the according confusion matrices share the same characteristics. With this sharing of characteristics is meant that the order of classes that can best be discriminated is the same: if class X has a much higher percentage of correctly classified patients than class Y for the original set, the same difference also occurs for the validation set. This observation can not be made for all classes when discriminating between five classes simultaneously, but it does apply to the majority of these classes as well as to all partial problems. Although it is rather hazardous to draw any conclusions on the basis of the above observations, we see reason enough to assume that the data that we have generated extends the limited original data in a way that is reasonably fair: the new data is not completely independent of the original data, but this dependency is limited.

The performance of discriminant analysis on both the original data using equal prior probabilities and resubstitution as well as the generated data is for all problems with only two classes reasonably good, but when five classes are to be discriminated simultaneously the performance enormously decreases. In the latter case the order in which the classes can best be discriminated is B, A, C/E and D, if the validation set is considered (matrix 4.46). Classification of patients belonging to class A and B is still acceptable, but the results for class D are dramatically bad. We therefore conclude that classes A and B are situated relatively isolated in the observation space, while the other classes and especially class D are much more spread. Since this is not a problem of discriminant analysis, but intrinsic to the data set, we expect the same results when using neural networks.

For comparison with neural networks — see section 7.2 for a comparison of all classification methods used — not only the classification percentages but also the computation time is of importance. When discriminant analysis is applied to the original data set containing all five classes and the time consuming cross-validation method was applied, a VAX 6000-410 needs only 34.5 seconds. When, again for the five-class problem, the generated training set containing 2500 patients is used to calibrate the observation space, after which the entire test set — also containing 2500 patients — is classified, the VAX 6000-410 needs no more than 33.5 seconds.

Chapter 5

Back-propagation

This chapter deals with back-propagation, a supervised method to train feed-forward neural networks. In the first part of this chapter the theory behind back-propagation is explained. In the second part the results obtained with this method are presented. A good introduction to this subject can be found in [21], while a more mathematical approach is taken in [22].

5.1 Network architecture

In the early days of neural networks research a well-known model was the perceptron. Because the perceptron consisted of a single layer of adaptable weights, only linear separable problems could be solved. Since many simple problems, the xor-function for instance, are not linearly solvable, a lot of effort was made to extend the perceptron to a multi-layer network, which does not suffer of this shortcoming. It was not until 1986 that the theory behind such a network was presented to a wide readership by Rumelhart and McClelland. This new neural network was called the *back-propagation network*.

The back-propagation network consists of two or more layers of processing units or *neurons*. Usually every neuron in a layer is connected to all neurons in the next layer, but this is no requirement. Because the back-propagation network is a *feed-forward network*, no connections exist between neurons in the same layer, nor between neurons of layers that are not successive. The first layer is called the *input layer*, the last layer the *output layer*, and all intermediate layers *hidden layers*. An example back-propagation network can be found in figure 5.1. The hidden layers act as feature detectors, which means that the layer combines all outputs of its previous layer to find certain relations in it.

This architecture is very similar to some biological neural networks, with the cell-body being represented by the processing unit, the dendrites by the incoming connections, and the axon by the outgoing connections. In biological neural networks information is recorded in the number and strength of the *synapses*, which are the connections between the neurons, while in a back-propagation network only the strength of the connections or *weights* is of importance. This distributive storage method has the advantage that several connections can be damaged without losing all stored information.

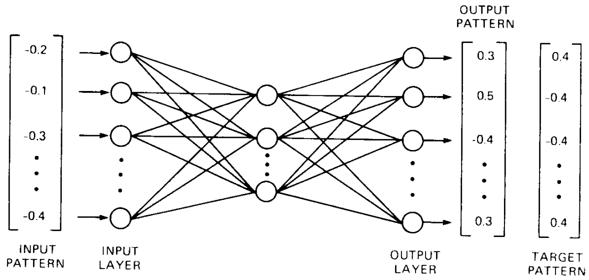


Figure 5.1: A possible back-propagation network.

5.2 Principles of the method

Teaching a pattern to a back-propagation network consists of two stages. In the first stage, covered in section 5.2.1, the input pattern is presented to the input layer and propagated forward through the network. In the second stage, described in section 5.2.2, the weights are adjusted so that the next time the same pattern is presented, the response of the network will resemble the desired output represented by the *target pattern* more closely.

5.2.1 Forward propagation

Forward propagation of an input pattern starts with the presentation of that pattern to the input layer. This means no more than copying each entry of the input pattern to the corresponding neuron of the input layer. For other layers than the input layer the value of a neuron, called the *activation*, is computed using the function

$$a_j = f\left(\sum_i a_i w_{ji}\right) \tag{5.1}$$

where i is the index of a neuron of layer n and j the index of a neuron of layer $n+1$. The symbol w_{ji} represents the weight between neurons i and j, and f is the *activation function*.

For the activation function several functions can be used, depending on the format of the patterns. For discrete valued patterns usually the step function is used, but when dealing with continuous valued patterns the sigmoid function

$$f(x) = \frac{1}{1+e^{-x}} \tag{5.2}$$

with derivative $f'(x) = f(x)(1-f(x))$ — see figure 5.2 — or the hyperbolic tangent function

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.3}$$

with derivative $f'(x) = 1 - f^2(x)$ is used — see figure 5.3 — although any continuous function for which the first-order derivative exists can serve as activation function.

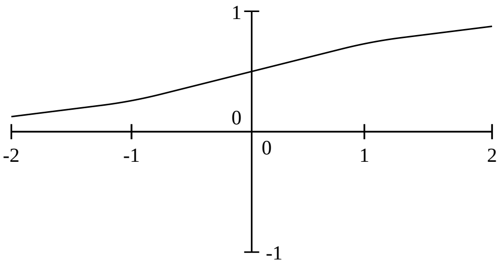


Figure 5.2: The sigmoid function.

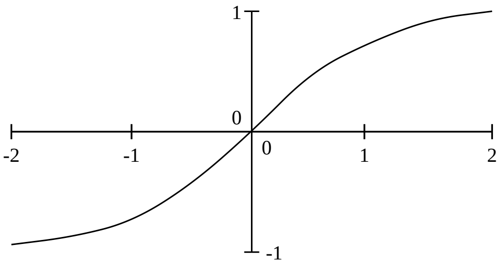


Figure 5.3: The hyperbolic tangent.

The perceptron makes use of a threshold, which is a value that is subtracted from the weighted sum of activations before the activation function is applied. Without a threshold the and-function can not be solved using only two inputs, since the separating line has to go through the origin; with a threshold however it can be solved. In a back-propagation network the threshold can be simulated by adding to every layer except the last one a neuron with a constant activation of -1. The activation of such a neuron — often called a *bias neuron* — never changes, so there is no need for it to be connected to neurons of the previous layer.

5.2.2 Backward propagation

After a pattern is completely propagated forward, the neurons of the output layer have a certain activation. As mentioned earlier, back-propagation is a supervised method, meaning that with each input pattern a target pattern must be supplied for training purposes. This target pattern, which represents the desired output, is compared with the pattern of the output layer to determine how well the neural network is trained. The fitness of a network is expressed using the following error function

$$E(w) = \frac{1}{2}\sum_p \sum_i (t_{p,i} - o_{p,i})^2 \tag{5.4}$$

In this equation w stands for all weights of the network, $t_{p,i}$ represents the i-th entry in the target vector of pattern p, and with $o_{p,i}$ the corresponding neuron of the output layer after forward propagation of pattern p is meant.

The best solution to a problem is reached if the error function is at its (global) minimum. With back-propagation this minimum is searched for using gradient descent for adapting the weights:

$$\Delta w_{ji} = -\eta \frac{dE(w)}{dw_{ji}} \tag{5.5}$$

Here η represents the *learning rate*, a constant that determines the speed of the training process. The value of η must be chosen carefully, for learning will be very slow if η is small, while oscillation can occur if η is too large.

Using the gradient descent rule it can be derived that for the hidden-to-output connections holds that

$$\Delta w_{lk} = \eta \sum_p a_k \delta_{p,l} \quad \text{with} \quad \delta_{p,l} = f'\left(\sum_i a_i w_{li}\right)(t_{p,l} - o_{p,l}) \tag{5.6}$$

In this equation j and l are indices to neurons of the last layer, and i and k refer to neurons of the previous layer. For the interested reader the proof of equation 5.6 is presented below.

The first step in the proof of equation 5.6 is writing out the definition of $\frac{dE(w)}{dw_{lk}}$, resulting in

$$\lim_{\Delta w_{lk} \to 0} \frac{1}{2} \sum_p \frac{\sum_{j \neq l}\left(t_{p,j} - f\left(\sum_i a_i w_{ji}\right)\right)^2 + \left(t_{p,l} - f\left(\sum_i a_i w_{li} + a_k \Delta w_{lk}\right)\right)^2 - \sum_j \left(t_{p,j} - f\left(\sum_i a_i w_{ji}\right)\right)^2}{\Delta w_{lk}}$$

After striking the terms that cancel each other remains

$$\lim_{\Delta w_{lk} \to 0} \frac{1}{2} \sum_p \frac{\left(t_{p,l} - f\left(\sum_i a_i w_{li} + a_k \Delta w_{lk}\right)\right)^2 - \left(t_{p,l} - f\left(\sum_i a_i w_{li}\right)\right)^2}{\Delta w_{lk}}$$

Working out of the quadratic terms results in

$$\lim_{\Delta w_{lk} \to 0} \frac{1}{2} \sum_p \frac{t_{p,l}^2 - 2t_{p,l} f\left(\sum_i a_i w_{li} + a_k \Delta w_{lk}\right) + \left(f\left(\sum_i a_i w_{li} + a_k \Delta w_{lk}\right)\right)^2 - t_{p,l}^2 + 2t_{p,l} f\left(\sum_i a_i w_{li}\right) - \left(f\left(\sum_i a_i w_{li}\right)\right)^2}{\Delta w_{lk}}$$

Striking $t_{p,l}^2$ and $-t_{p,l}^2$ that cancel each other, and cleverly rearranging the other terms gives

$$\lim_{\Delta w_{lk} \to 0} \frac{1}{2} \sum_p \frac{-\left(f\left(\sum_i a_i w_{li} + a_k \Delta w_{lk}\right) - f\left(\sum_i a_i w_{li}\right)\right)}{\Delta w_{lk}} \left(2t_{p,l} - f\left(\sum_i a_i w_{li} + a_k \Delta w_{lk}\right) - f\left(\sum_i a_i w_{li}\right)\right)$$

Application of the definition of the first-order derivative of f and working out the limit results in

$$-\frac{1}{2} \sum_p a_k f'\left(\sum_i a_i w_{li}\right)\left(2t_{p,l} - 2f\left(\sum_i a_i w_{li}\right)\right)$$

Multiplication with $-\eta$ and changing to the notation of equation 5.6 gives the desired result

$$\Delta w_{lk} = \eta \sum_p a_k \delta_{p,l} \quad \text{with} \quad \delta_{p,l} = f'\left(\sum_i a_i w_{li}\right)(t_{p,l} - o_{p,l})$$

In a similar way with application of the chain rule for differentiation it can be derived that for all other connections

$$\Delta w_{lk} = \eta \sum_p a_k \delta_{p,l} \quad \text{with} \quad \delta_{p,l} = f'\left(\sum_i a_i w_{li}\right) \sum_i \delta_{p,i} w_{li} \tag{5.7}$$

holds. Again j and l are indices to neurons of an arbitrary layer except the last, and i and k refer to neurons of the previous layer.

From equation 5.6 it is known how the δ's for the neurons of the output layer can be computed; the δ's for the other neurons can be computed if the principle described in equation 5.7 is reversed:

$$\delta_{p,k} = f'\left(\sum_i a_i w_{ki}\right) \sum_j \delta_{p,j} w_{jk} \tag{5.8}$$

Notice that in this equation k is an index of a neuron in a certain layer, i refers to neurons in the previous and j to neurons in the next layer. The δ's determined in the last layer — and thus indirectly the error — are propagated backward through the network to compute the δ's of the other layers. It is this mechanism that has provided the name (error) back-propagation for this type of neural networks.

Equations 5.6 and 5.7 suggest that updating of the weights is done after all the patterns have been presented to the network. Of course this strategy, called *learning by epoch*, can be applied, but in [22, p. 119] can be found that updating after each pattern, *learning by pattern*, can be more effective. The reason for this is that with learning by pattern, the patterns can be presented in random order, making the path through the weight-space stochastic. The learning rate in this case must be small to avoid oscillating of the network. Although the effectiveness of the two approaches depends on the problem, learning by pattern usually prevails, especially if the training set is very regular or redundant.

5.3 Variations on the basic algorithm

Many variations on the basic back-propagation algorithm exist, all serving different purposes. In sections 5.3.1 to 5.3.3 some of these variations are discussed.

5.3.1 Noise

As explained in section 5.2.2, training a back-propagation network is done by minimizing an error function. Starting with a random set of weights gradient descent is used to find the global minimum. But it is possible that the network gets stuck in a local minimum that is not the optimum. Such a minimum occurs if two or more errors compensate each other. The basic back-propagation algorithm can not get out of a local minimum, unless the algorithm is slightly changed.

Apart from methods that change the architecture of the network, two more or less similar methods are in use, both utilizing noise during training. The first method adds noise to the training patterns. This distortion changes the topology of the error function, making it possible that the network is no longer situated in the local minimum of before.

The other method adds noise to the weights of the network. This way not the topology of the error function is changed, but the position of the network in this landscape. Again it is possible that after this distortion the network has come out of the local minimum.

The amount of noise that must be added to get the network out of a local minimum depends on the distance between the pit of that minimum and the lowest top surrounding it. Usually these relative depths in the error surface are not known, so a lot of experimenting has to be done to determine the right amount of noise. If the amount is too small, the network can never get out of a local minimum, but if too much noise is added, the features in the data will no longer be recognizable, resulting in a problem that will never be solved.

5.3.2 Momentum

As mentioned before learning can be very slow if the learning rate (η) is small. Instead of increasing η, which could cause oscillation of the network, the back-propagation algorithm can be extended with a *momentum term*. This alters the procedure according to which the weights are changed to

$$\Delta w_{ji}(t+1) = -\eta \frac{dE(w)}{dw_{ji}} + \alpha \Delta w_{ji}(t) \tag{5.9}$$

where t represents the time the network is updated, and α is a constant between 0 and 1, determining the magnitude of the momentum.

The idea behind the momentum term is to provide each connection with a certain amount of inertia. This approach is especially useful for error-surface valleys with steep sides and a shallow slope along the bottom, in which normal back-propagation is very slow. Another advantage of the momentum term is that oscillation is reduced, because a negative change of a weight due to the gradient term is partially compensated by the momentum term of the previous positive change.

5.3.3 Weight decay

When a neural network is trained to solve a classification problem, a — hopefully — representative training set is presented to the network. Using the principles described in section 5.2.1 and 5.2.2, the network is optimized for this set. If the problem is solved well, data from the test set can be classified equally well as data from the training set. In this case the network is said to have a good ability for *memorization*, which is the performance on the training data, as well as for *generalization*, being the performance on the test data.

It is however possible that at some point in the training process the network starts to tune itself to certain relations in the training set that are not features of problem. An example of over-learning can be found in figures 5.4 and 5.5. This phenomenon is called *over-learning*, and is something that should be avoided. Over-learning can be detected using a test set: if the performance of the network on the test set starts to decrease, while the performance on the training set still increases, over-learning occurs and training should be stopped.

An approach to prevent over-learning is *weight decay*, a method in which the connections are given a tendency to decay to zero, so that a connection virtually disappears unless it is reinforced. Relations in the training set that are not features of the problem, are usually relatively weak compared to real features. The impact of these relations on the connections will therefore be small in basic back-propagation, but will be nil in the case of weight decay, improving generalization.

Weight decay is achieved by extending the error function with a penalty term to

$$E(w) = \frac{1}{2} \sum_p \sum_i (t_{p,i} - o_{p,i})^2 + \frac{\gamma}{2} \sum_j w_j^2 \tag{5.10}$$

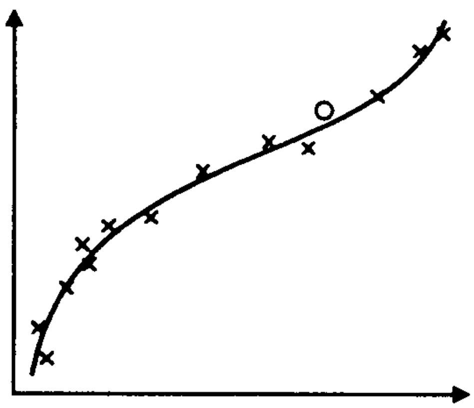


Figure 5.4: The network, which is trained using the x's, approximates the circel well.

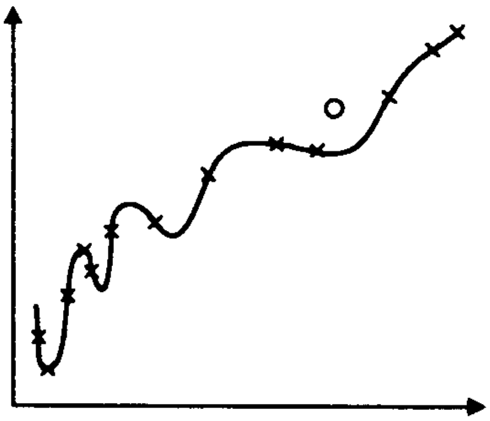


Figure 5.5: This network has a smaller error for the x's, but a worse approximation of the circle.

with j referring to all weights of the network. Analogous to equation 5.7 it can be derived that the amount a weight is updated now changes to

$$\Delta w_{lk} = \eta \left(\sum_p a_k \delta_{p,l} - \gamma w_{lk} \right) \tag{5.11}$$

In these equations γ is a constant determining the magnitude of the decay term. The parameter γ must be chosen with care, for a too small value does not prevent over-learning, and a value that is too large will prevent the network from learning the problem. Experimenting is the only solution to this dilemma.

A disadvantage of this method is that it forces large weights proportionally to zero, while these values could well be needed to solve the problem. To meet this shortcoming different penalty terms have been invented. One of these methods that seems to be promising is soft weight-sharing. This technique tries to distribute the weights using several Gaussian distributions, each with a different mean and standard deviation. A well-readable explanation of this can be found in [23].

5.4 Results

To make the data suitable for back-propagation, the representation of the classes has to be slightly altered. Instead of a simple number back-propagation needs a target pattern that represents the class of the input pattern. When dealing with n classes, the target pattern is a vector consisting of n elements of which $n-1$ elements have, in our case, the value 0.1, and just one element has the value 0.9. The index of the single element in this vector represents the class of the corresponding input pattern. The above operation on the data is combined with the scaling operation, and is performed by the program *format* described in appendix A.

A short users-manual for the back-propagation implementation used — *bp* — can be found in appendix A as well.

5.4.1 Measuring the performance

In section 5.2.2 is explained that the principle of back-propagation is derived by minimizing the error or cost function of equation 5.4, representing the *total-sum-of-squares* or *TSS*. A disadvantage of this function as a measure for the performance is that it depends on the number of neurons in the output layer, but also on the number of patterns. Therefore often a similar function is used, representing the *root-mean-squares*

$$RMS(w) = \sqrt{\frac{\sum_p \sum_i (t_{p,i} - o_{p,i})^2}{n\_patients\ n\_outputs}} \tag{5.12}$$

where w represents all weights of the network, $t_{p,i}$ the i-th element in the target vector of pattern p, $o_{p,i}$ the corresponding neuron of the output layer after forward propagation of pattern p, and $n\_patients$ and $n\_outputs$ the number of patients respectively neurons in the output layer.

The error function of equation 5.12 can of course be used to measure the performance of the network, but it is not exactly what we are interested in. Since we are more interested in the fraction of correct or incorrect classification, we have designed a new error function *ICF* described by

$$ICF(w) = \sqrt{\frac{\sum_c \left(\frac{n\_patients_c - \sum_p \Phi(t_p, o_p)}{n\_patients_c}\right)^2}{n\_classes}} \tag{5.13}$$

In this equation c is an index over all classes, $n\_classes$ is the number of classes, p is an index over all patients in the class under observation, and $n\_patients_c$ is the number of patients in class c. The symbol Φ stands for a function that is 1 if the class represented by t_p coincides with the class indicated by o_p, and is 0 otherwise. The class that vector o_p represents is determined by the element of o_p that has the highest value.

The term $n\_patients_c - \sum_p \Phi(t_p, o_p)$ is in fact nothing more than the number of erroneously classified patients in a certain class, so in other words *ICF* is the square root of the mean squared error fraction. The method of squaring the error fraction after which the root is taken is chosen instead of averaging over the error fractions, because we want to favour for example the (0.3, 0.3) solution over the (0.1, 0.5) solution in case of two classes. This way an extremely bad classification of a certain class has more influence on the measure of performance than when simply averaging over the error fractions.

The error function bears a direct relation to the confusion matrix that can be obtained with discriminant analysis, since this matrix displays all classification percentages from which the error fractions can be easily derived. Furthermore the same error function can be used when dealing with other neural networks,

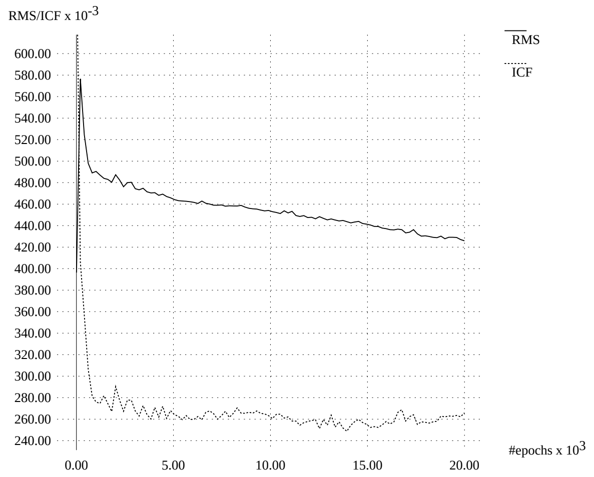


Figure 5.6: Comparison between the *RMS* and *ICF* for a back-propagation network.

like Kohonen's self-organizing feature map, so a direct comparison of the different methods is possible.

Because in *ICF* all classes are of equal importance, minimizing *ICF* coincides with maximizing the correct classification per class. The consequences of this choice are that the number of patterns in the training set must approximately be equal per class to prevent the network from focusing on the largest class, and secondly that, when comparing back-propagation with discriminant analysis, equal prior probabilities must be assigned to each class.

In contrast to many implementations where the error function is computed *during* a learning epoch, we compute it *after* each learning epoch. Although this approach is a bit less efficient, it is more fair when learning by pattern, because if the performance of the network on a pattern is measured immediately after adjusting the network to it, the performance will be good, while the next patterns that are presented to the network will partially cancel this result. For this reason it is better to freeze a network before evaluating it.

Unfortunately, as can be seen in figure 5.6, the *RMS* and *ICF* are not always fully correlated. This means that the network does not try to maximize the fraction of correct classification, something that we would like it to do. Another observation that can be made is that the *ICF* is much more jumpy than the *RMS*. The reason for this behaviour is that for the *ICF* classification of a pattern is correct or incorrect, there is no intermediate possibility. If the weights of the network are slightly adapted, resulting in only a small change of the *RMS*, it is possible that for some patterns a different neuron of the output layer becomes the winner, thus resulting in a different classification. For the *ICF* classification of these patterns now changes radically, from correct to incorrect or vice versa. The jumpy behaviour can partially be resolved by averaging the *ICF* over a number of epochs.

5.4.2 Approach to solving the problem

Just like with discriminant analysis, we first try to discriminate between healthy people — class A — and every disease separately, before proceeding with the more difficult problem of discriminating between all five classes at the same time. The experience gained from examining these problems is used to get a feeling for the problem domain and to avoid doing superfluous work.

A back-propagation network has many parameters that have to be adjusted correctly to get a good performance. Some of these parameters have a big influence on the performance, others are of lesser importance. To find the optimal network all combinations of the parameters should be tried, but this method takes a lot of time. The approach we therefore take is determining the optimal values for the parameters of the basic algorithm, extended with a momentum term for speed, in order of decreasing influence. For the parameters that are not determined yet values are assumed based on rules of thumb or previous experience. After the — hopefully — optimal network is determined, two variations on the basic algorithm — adding of noise to the learn patterns and weight decay — are tried to see if the performance improves.

During training the performance of the network on the training set is monitored using the *RMS* function. The reason for monitoring the training set is that we want to see if the network converges well, but verification of several theoretical propositions — the learning rate influences the speed of learning for instance — is now possible as well. For these purposes the *RMS* is better suited than the *ICF*, because the *RMS* is the error measure the network tries to minimize[1], while the *ICF* is a measure the network is completely unaware of. Furthermore we are not interested in the classification results of the network on the training set, because on this set 100% correct classification can easily be achieved if only the size of the network is large enough [22, p. 142]. To reduce the number of points in the figures to 20 without risking that one of the selected points is non-representative for the performance curve for each point the average of 25 successive points is taken.

The performance of the network on the test set is also monitored during training, but with this set the *ICF* function is used. Here monitoring is done to check on the generalization capabilities of the network, something that is very important when dealing with classification problems. Although generalization and detection of over-learning — see section 5.3.3 — could probably be done at least as well using the *RMS*, we have chosen for the *ICF* because it displays the fraction of incorrect classification as well, and this is something *we* want to minimize. Since from the *RMS* the ability to classify apparently can not be determined, which can be seen in figure 5.6, but over-learning probably can be detected using classification fractions, the *ICF* seems to be the only logical choice. Again, to limit the number of points in a figure to 20 and to control the jumpy behaviour of the *ICF* each point represents the average of 25 successive points.

The decision to adjust a parameter to a particular value is primarily based on the performance of the network on the test set. Only if these figures do not

[1] Actually the network tries to minimize the *TSS*, but the *RMS* is fully correlated with it.

differ enough to make a clear choice for a particular value of a parameter, the performance on the training set is taken into consideration. A better or faster convergence could be a reason to favour a particular value.

If a network seems to be promising, the validation set is presented to it. From the resulting figure the number of epochs is determined after which the network classifies best. The classification results the network achieves in this stage are used to draw up a confusion matrix, which can be used for comparison with other types of neural networks or classifying techniques.

In the remainder of this section the parameters in order of expected decreasing influence on the performance of a back-propagation network are briefly discussed to motivate their earlier mentioned initial adjustments.

Architecture

The architecture of a neural network is the degree of freedom that has the biggest influence on the performance. Although this parameter is the first to be varied and thereby no initial value has to be chosen, some annotations have to be made.

First it has to be remarked that the only aspects of the architecture that can be varied are the number of hidden layers and the number of neurons per layer. The network has to stay a feed-forward network. Although there is no restriction on the number of hidden layers, in [22, p. 142] can be found that a network with three or more hidden layers offers no additional computational power over a network with only two hidden layers. In view of the fact that, the more layers a network consists of, the less well it can be parallelized, it seems to be sensible to confine the architecture to one or two hidden layers.

As mentioned before, also in [22, p. 142] can be found that a function can be approximated to any given accuracy, if only the size of the network is large enough. For a classification problem this means that a correct classification of the training set up to 100% can be achieved. Unfortunately the performance on the test set will probably be very poor, because the network has overfitted the training set. This is what was called over-learning in section 5.3.3.

Learning rate

Although some people use the rule of thumb that in case of learning by pattern the learning rate in a back-propagation network should approximately be equal to 0.25 divided by the number of patterns in the training set, our experience with the same data is that a larger value can be chosen without the risk of severe oscillation. Therefore the initial learning rate (η) is set at 0.01, by which we hope to limit the number of epochs needed.

Momentum

The momentum term — see section 5.3.2 — is not a part of the basic back-propagation algorithm, but we use it anyway because of the acceleration it gives to the training process. If there is only one — global — minimum, the momentum term has no influence on the performance of the network. If however several minima exist, the momentum term could cause the network to get stuck in one of them which it otherwise would have avoided. But just as well could as a result of the momentum term a local minimum be avoided where otherwise the network would get stuck in.

The constant that determines the magnitude of the momentum term (α) must lie between 0 and 1 and is usually chosen relatively high [22, p. 123]. As an initial value for α we therefore choose 0.8.

Initial weights

When a neural network is initialized, random values are assigned to its weights. Since every set of weights defines a position on the surface of the error function, the initial weights can influence the performance if this surface contains local minima. Therefore the probability distribution from which the weights are drawn is of importance. However, to avoid saturation of the activation function, which will be explained in more detail in the next section about the activation function, the initial weights must meet certain requirements. Therefore the initial weights are drawn from an uniform distribution over the interval [-0.5,0.5]. Only if the performance curve of the training set suggests that the network has got stuck in a local minimum, a different initialization of the weights is tried.

Activation function

In section 5.2.1 two activation functions for continuous valued patterns were presented, namely the sigmoid and the hyperbolic tangent. Both functions have a first-order derivative that has the highest value in the interval [-1,1]; outside this interval the functions are relatively flat. If a variation in a pattern or a change of weights should have any effect on the output of a neuron, the weighted sum to which the activation function is applied should therefore not lie too far outside [-1,1]. It is for this reason that the data is scaled between 0.1 and 0.9, and that the initial weights are drawn from the interval [-0.5,0.5]. The data is not scaled between -0.9 and 0.9 because this would cause some parameters to become 0, thus not contributing to the weighted sum anymore.

For the activation function only the sigmoid is considered, because this function is less flat outside the interval [-1,1] than the hyperbolic tangent. Another reason to use the sigmoid function is that it is more or less a standard when dealing with back-propagation networks, in contrast to the hyperbolic tangent which is used only incidentally. If the hyperbolic tangent is to be used, the target patterns of all sets have to be changed because the range of the hyperbolic tangent starts at -1 instead of 0. To have full advantage of this larger interval it is wise to choose a value for the low value of the target pattern that is close to -1, usually -0.9 is taken.

Learning grain

At the end of section 5.2.2 we have explained that updating of the weights, which is neural network equivalent of learning, can take place after the presentation of a single pattern or all patterns of the training set. This difference is called the *grain* of the training process. Since in [22, p. 119] can be found that learning by pattern in combination with a random order of presenting the patterns is useful for avoiding local minima, only this fine learn grain is considered.

5.4.3 Discrimination between A and B

In this section the results of back-propagation applied to the problem of discriminating between classes A and B are presented. The training, test and validation set are described in section 3.5, while a discussion of the results can be found in section 5.5.

Varying the architecture

Eight different network architectures are examined, of which four have only one hidden layer and four have two hidden layers. The performance of the networks on the training set can be found in figures 5.7 to 5.8, while the figures 5.9 to 5.10 show the results on the test set.

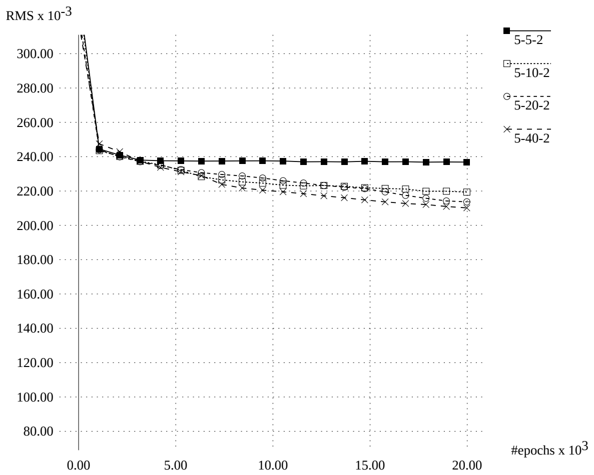


Figure 5.7: Performance on the AB-training set for several three-layer architectures.

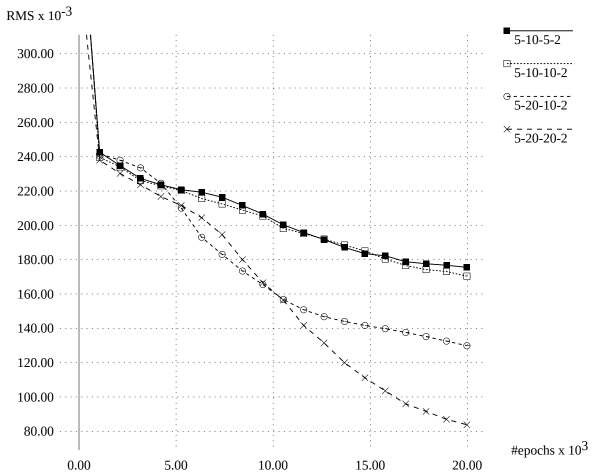


Figure 5.8: Performance on the AB-training set for several four-layer architectures.

In figures 5.9 and 5.10 can be observed that most architectures have the best performance on the test set during the first 7500 epochs. To reduce the computation time needed for each simulation the number of epochs is limited to 7500 in the remainder of this section. In this interval the network with one hidden layer containing 40 neurons gives the best performance and is used for further optimizing.

Varying the learning rate

Apart from a learning rate of 0.01, which was chosen as default in section 5.4.2, two other values are tried: 0.0005 and 0.2. The results on the training set

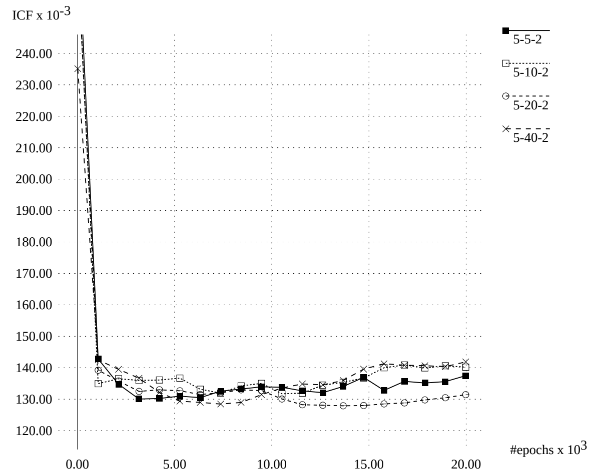


Figure 5.9: Performance on the AB-test set for several three-layer architectures.

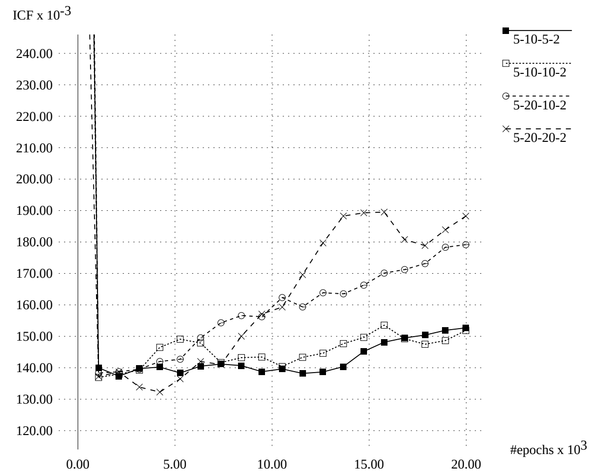


Figure 5.10: Performance on the AB-test set for several four-layer architectures.

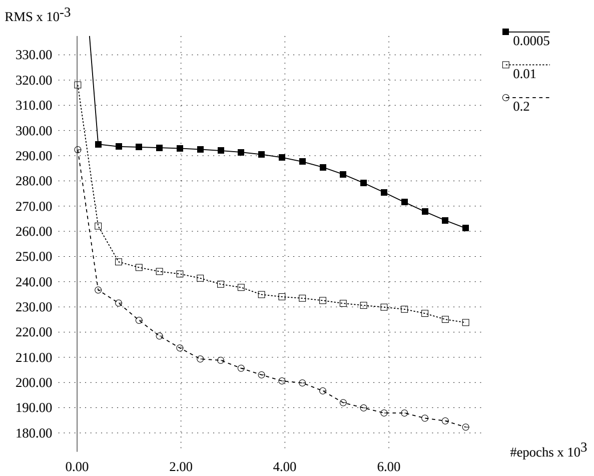


Figure 5.11: Performance on the AB-training set for several learning rates.

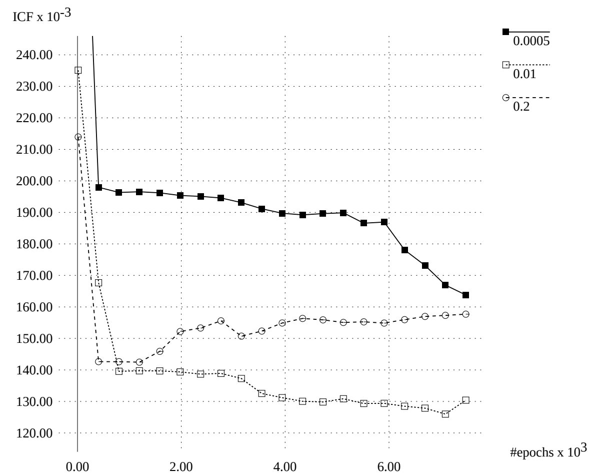


Figure 5.12: Performance on the AB-test set for several learning rates.

are shown in figure 5.11, and the performance on the test set is presented in figure 5.12.

A learning rate of 0.01 seems to give the best performance.

Varying the momentum term

The values of α, which determines the magnitude of the momentum term, that are tried are 0.00, 0.40, 0.80 and 0.95. Figure 5.13 shows the performance on the training set, while the performance on the test set can be found in figure 5.14.

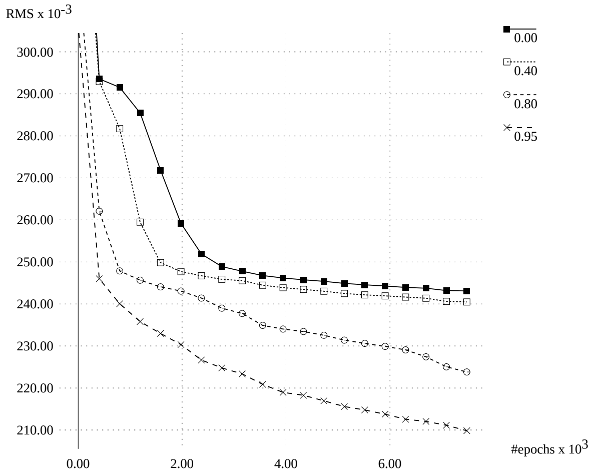


Figure 5.13: Performance on the AB-training set for several momenta.

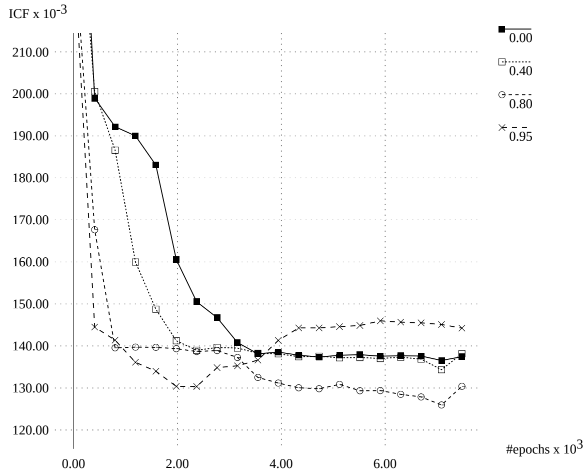


Figure 5.14: Performance on the AB-test set for several momenta.

The best results are achieved with an α of 0.80.

Adding noise

The results of adding noise to every pattern of the training set during the training process can be found in figure 5.15 for the training set, and in figure 5.16 for the test set. The maximum absolute noise that is added to a pattern is respectively 0.00, 0.02, 0.05 and 0.010.

This variation on the basic algorithm does not improve the performance of the network and is therefore left out of consideration.

Using weight decay

For the magnitude of the weight decay term, which is expressed by γ, the values 0.000, 0.001 and 0.010 are tried. The results of these simulations on the training set can be found in figure 5.17, while figure 5.18 displays the results on the test set.

Weight decay does not seem to improve the performance of the network, and is thus no longer considered in this section.

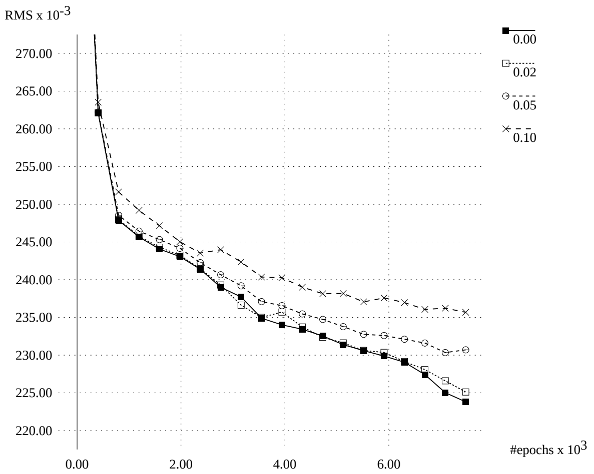


Figure 5.15: Performance on the AB-training set using noise.

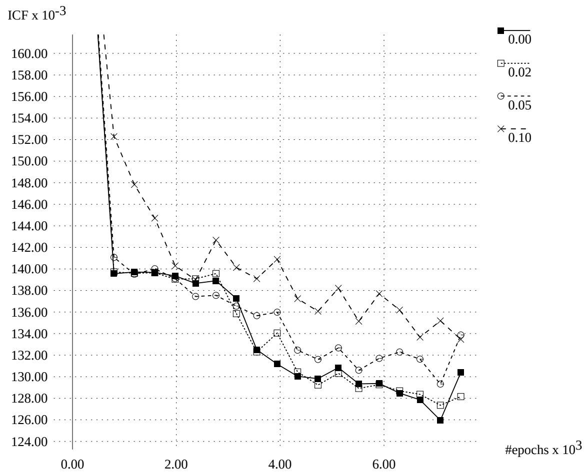


Figure 5.16: Performance on the AB-test set using noise.

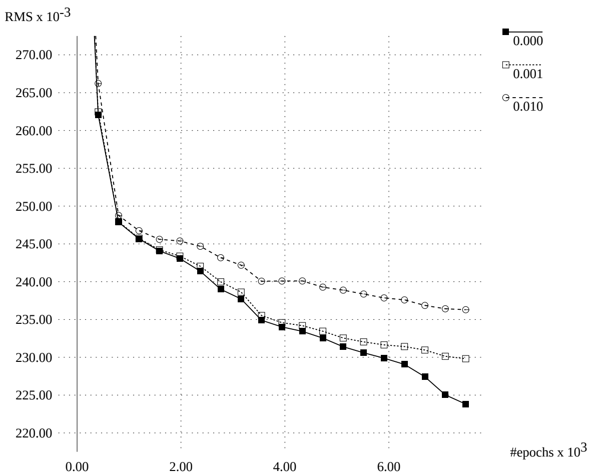


Figure 5.17: Performance on the AB-training set using weight decay.

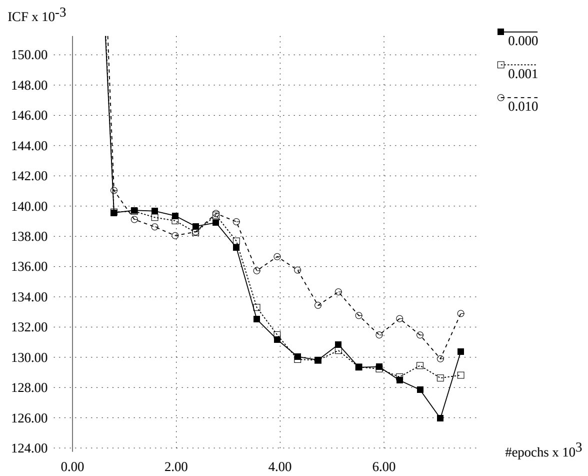


Figure 5.18: Performance on the AB-test set using weight decay.

Performance on the validation set

The performance on the validation set of the 5-40-2 network with a learning rate of 0.01, a momentum of 0.80 with neither noise nor weight decay is presented in figure 5.19.

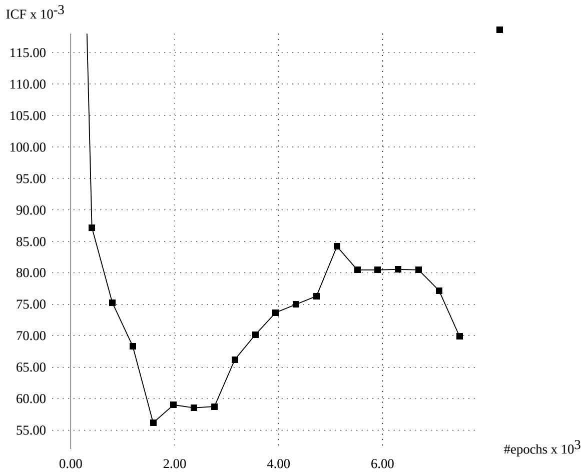


Figure 5.19: Performance on the AB-validation set for the optimal network.

The network gives the best performance after 1249 epochs; the classification percentages of this situation are given in confusion matrix 5.14.

$$\begin{array}{c} \\ A \\ \\ B \end{array} \begin{array}{c} \begin{array}{cc} A & B \end{array} \\ \begin{pmatrix} 93.8 & 6.2 \\ & \\ 0.0 & 100.0 \end{pmatrix} \end{array} \quad (5.14)$$

5.4.4 Discrimination between A and C

This section deals with the problem of discriminating between classes A and C using back-propagation. A discussion of the results can be found in section 5.5.

Varying the architecture

Again the eight architectures described in section 5.4.3 are examined resulting in figures 5.20 to 5.21, showing the performance on the training set, and figures 5.22 to 5.23, which show the results on the test set.

In the first 7500 epochs — to which the other simulations in this section will be limited — the network with 20 neurons in one hidden layer gives the best performance.

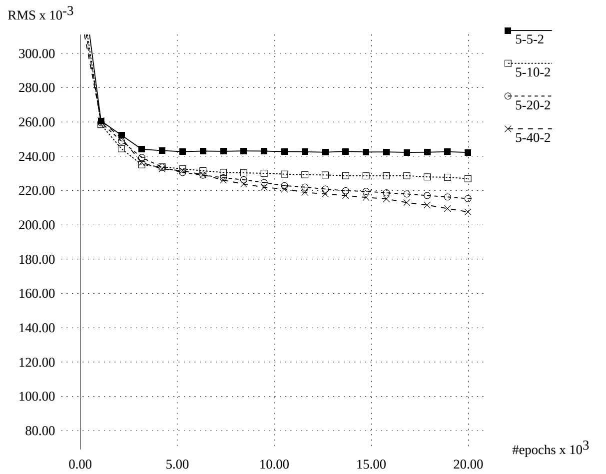


Figure 5.20: Performance on the AC-training set for several three-layer architectures.

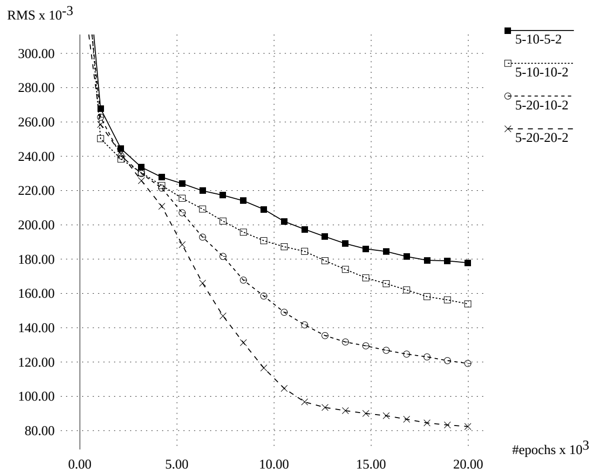


Figure 5.21: Performance on the AC-training set for several four-layer architectures.

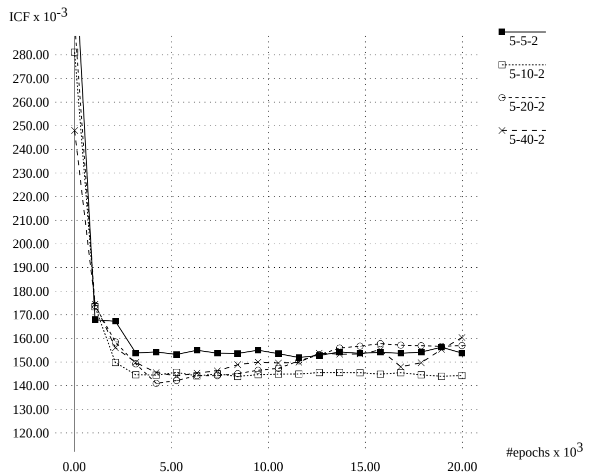


Figure 5.22: Performance on the AC-test set for several three-layer architectures.

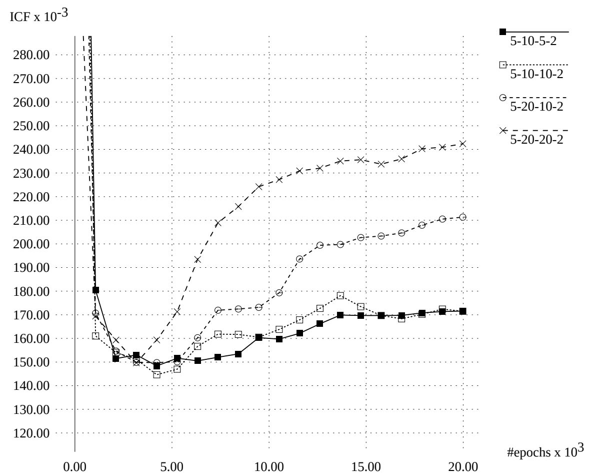


Figure 5.23: Performance on the AC-test set for several four-layer architectures.

Varying the learning rate

The results of variations in the learning rate can be found in figure 5.24 for the training set and in figure 5.25 for the test set.

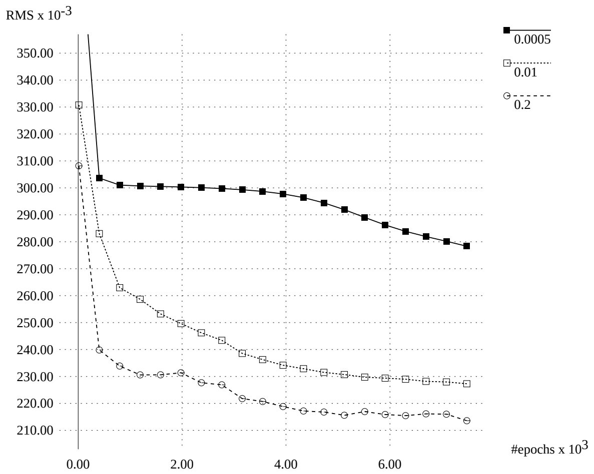


Figure 5.24: Performance on the AC-training set for several learning rates.

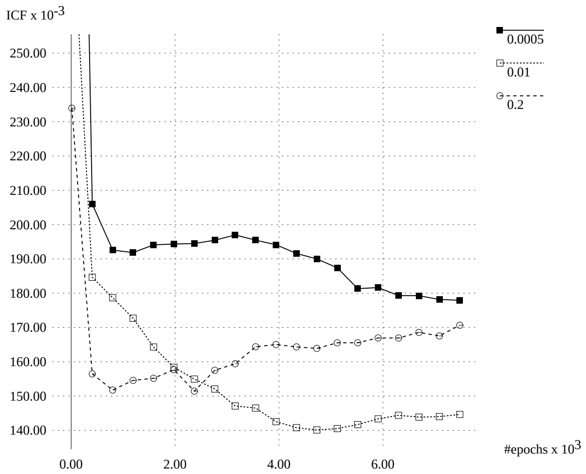


Figure 5.25: Performance on the AC-test set for several learning rates.

The best performance is achieved with a learning rate of 0.01.

Varying the momentum term

If the magnitude of the momentum term is varied in the same way as in section 5.4.3, the performance on the training and test set can be found in figure 5.26 respectively 5.27.

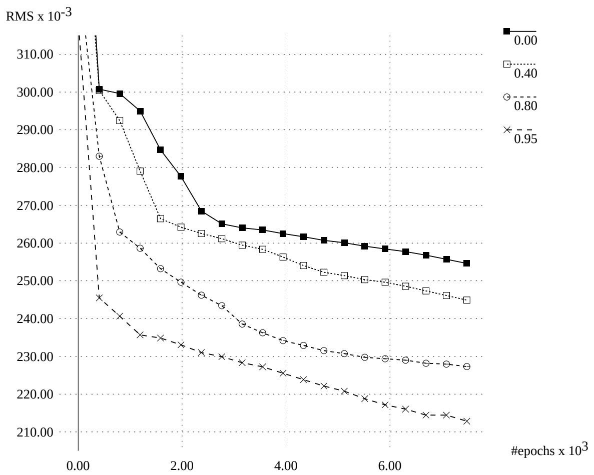


Figure 5.26: Performance on the AC-training set for several momenta.

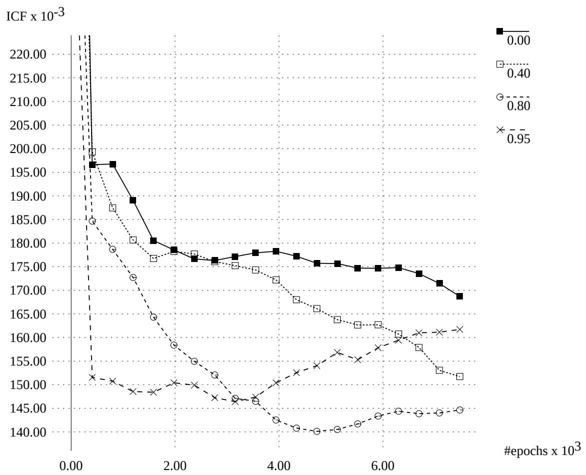


Figure 5.27: Performance on the AC-test set for several momenta.

The best choice for α seems to be 0.80.

Adding noise

Adding noise to the patterns of the training set gives the results shown in figures 5.28 and 5.29 for the training and test set.

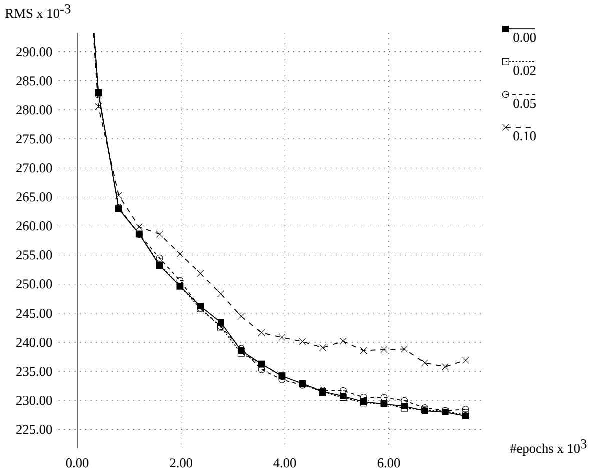


Figure 5.28: Performance on the AC-training set using noise.

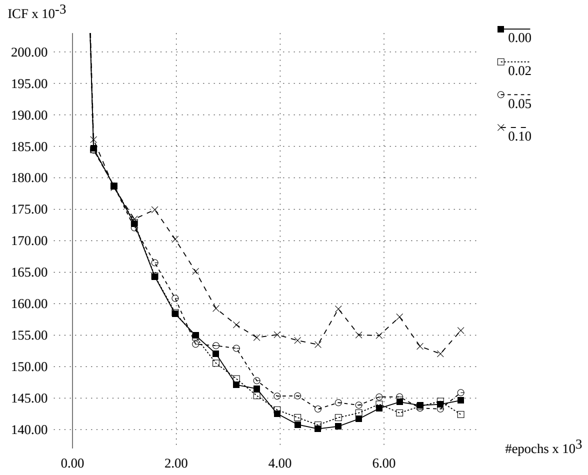


Figure 5.29: Performance on the AC-test set using noise.

For this problem using no noise at all seems to be the best solution.

Using weight decay

The extension of the basic algorithm with a weight decay term results in figure 5.30 which shows the performance on the training set, and figure 5.31 which shows the performance on the test set.

A small decay term with an γ of 0.001 gives a slight improvement of the performance and will therefore be used when determining the performance on the validation set.

Performance on the validation set

The results on the validation set of the 5-20-2 network together with a learning rate of 0.01, a momentum of 0.80 and a weigh decay term of 0.001 without the addition of noise can be found in figure 5.32.

The confusion matrix of the network after 5828 epochs is given in matrix 5.15.

$$\begin{array}{cc} & \begin{array}{cc} A & C \end{array} \\ \begin{array}{c} A \\ \\ C \end{array} & \left(\begin{array}{cc} 81.4 & 18.6 \\ & \\ 7.1 & 92.9 \end{array} \right) \end{array} \quad (5.15)$$

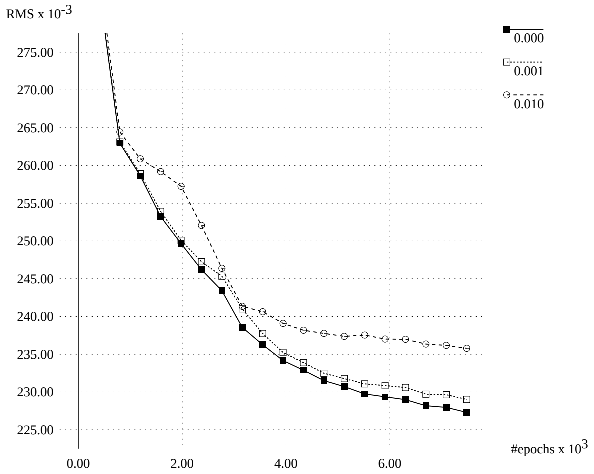


Figure 5.30: Performance on the AC-training set using weight decay.

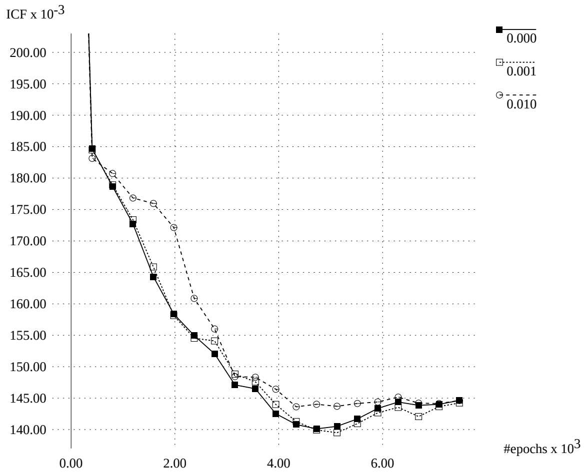


Figure 5.31: Performance on the AC-test set using weight decay.

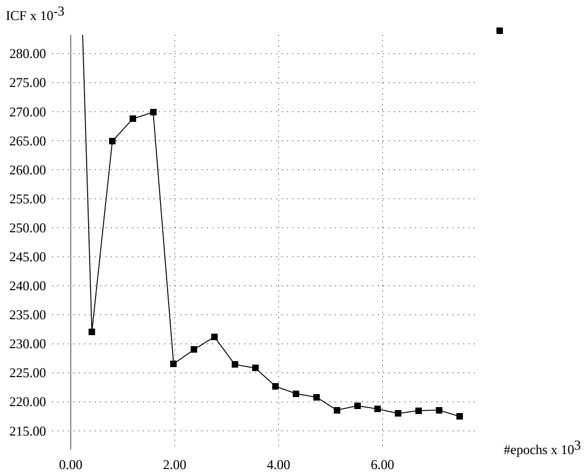


Figure 5.32: Performance on the AC-validation set for the optimal network.

5.4.5 Discrimination between A and D

The results of the use of back-propagation to discriminate between class A and D are given in this section. A discussion of these results can be found in section 5.5.

Varying the architecture

The performance of the previously described networks can be found in figures 5.33 to 5.36. Figures 5.33 and 5.34 show the performance on the training set, while in figures 5.35 and 5.36 the performance on the test set can be found.

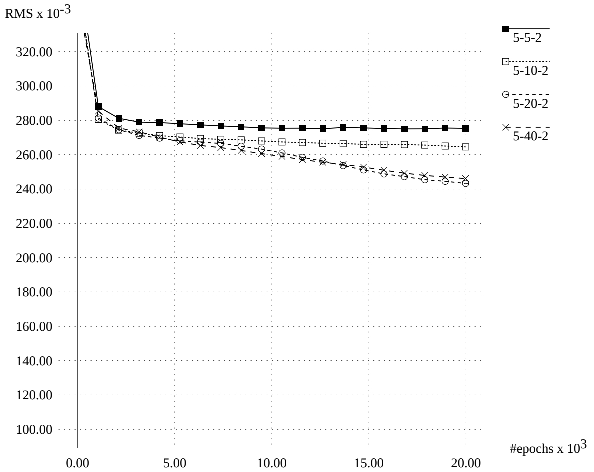


Figure 5.33: Performance on the AD-training set for several three-layer architectures.

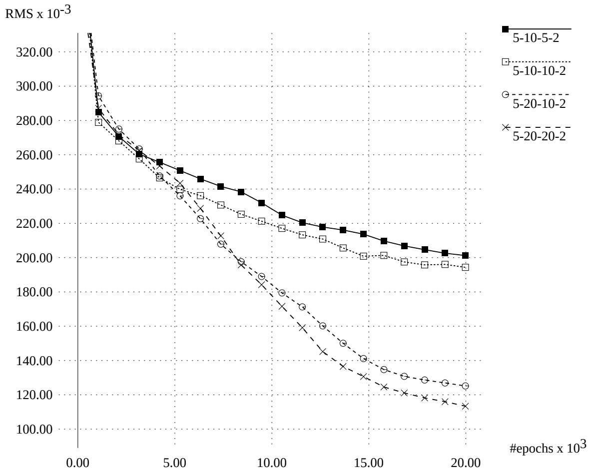


Figure 5.34: Performance on the AD-training set for several four-layer architectures.

The network with one hidden layer of 40 neurons gives the best results for the first 7500 epochs and is chosen as the default network for this problem.

Varying the learning rate

Variations in the learning rate result in figure 5.37 displaying the performance on the training set, and in figure 5.38 showing the performance on the test set.

A learning rate of 0.01 gives the best results.

Varying the momentum term

The influence of the momentum term on the performance of the network on the training set is shown in figure 5.39, while the influence on the test set is displayed in figure 5.40.

The best results are achieved with a momentum term of 0.80.

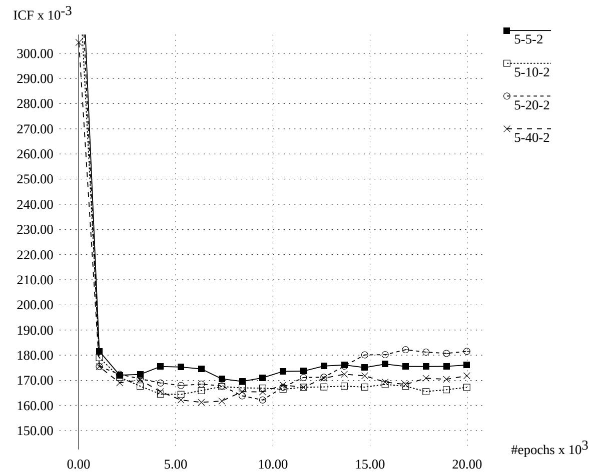


Figure 5.35: Performance on the AD-test set for several three-layer architectures.

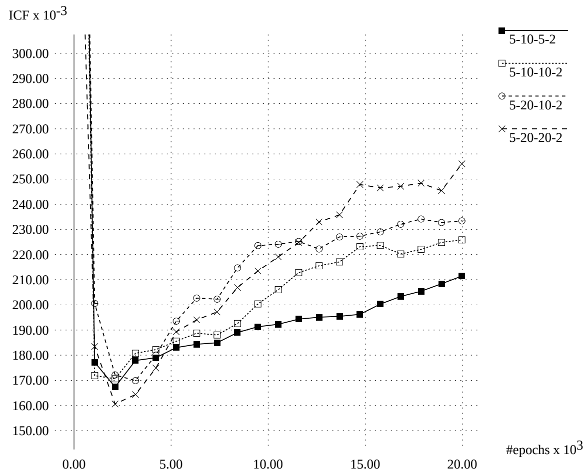


Figure 5.36: Performance on the AD-test set for several four-layer architectures.

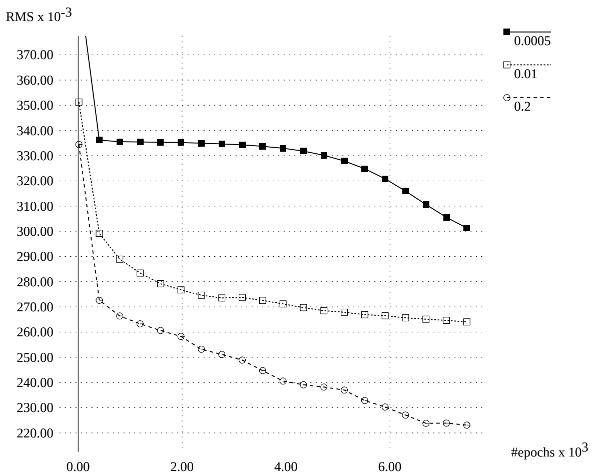


Figure 5.37: Performance on the AD-training set for several learning rates.

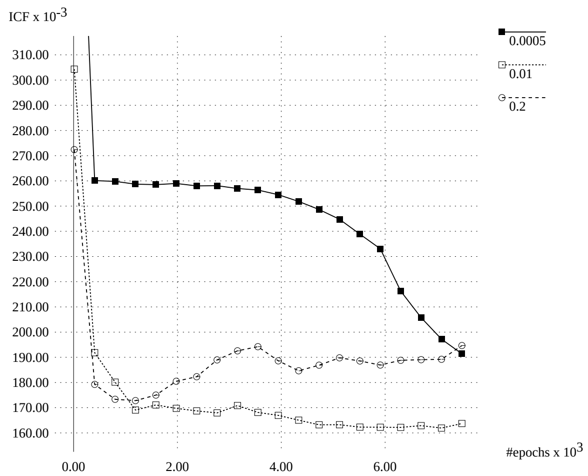


Figure 5.38: Performance on the AD-test set for several learning rates.

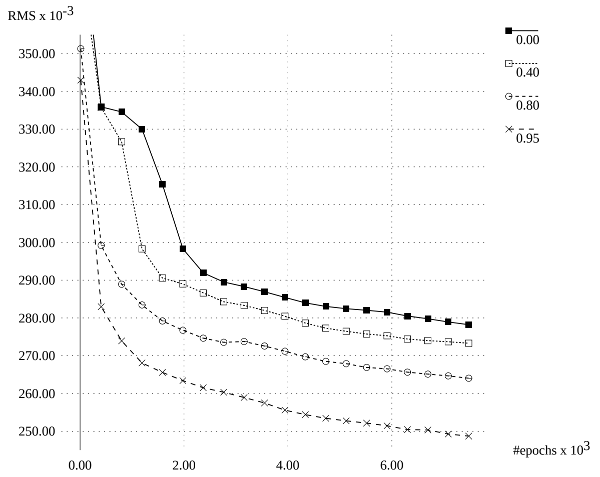


Figure 5.39: Performance on the AD-training set for several momenta.

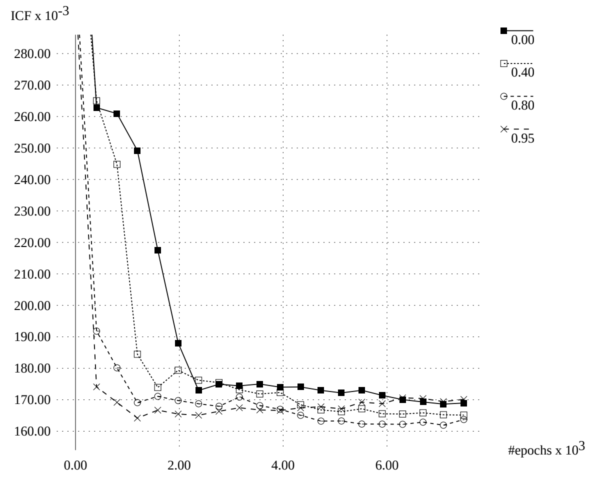


Figure 5.40: Performance on the AD-test set for several momenta.

Adding noise

If to each training pattern during the training stage a certain amount of noise is added, the results of back-propagation on the training and test set change to figures 5.41 respectively 5.42.

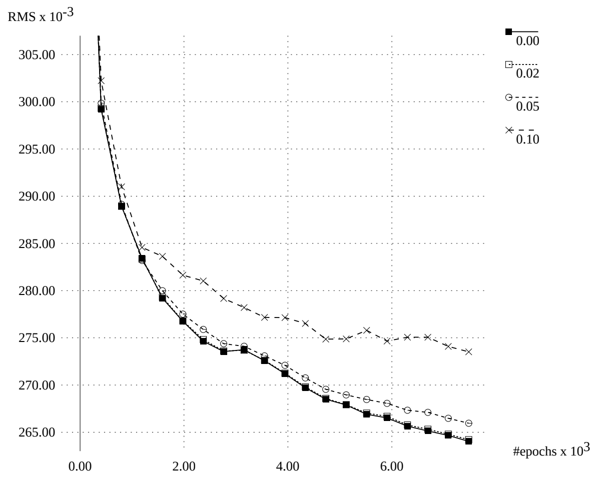


Figure 5.41: Performance on the AD-training set using noise.

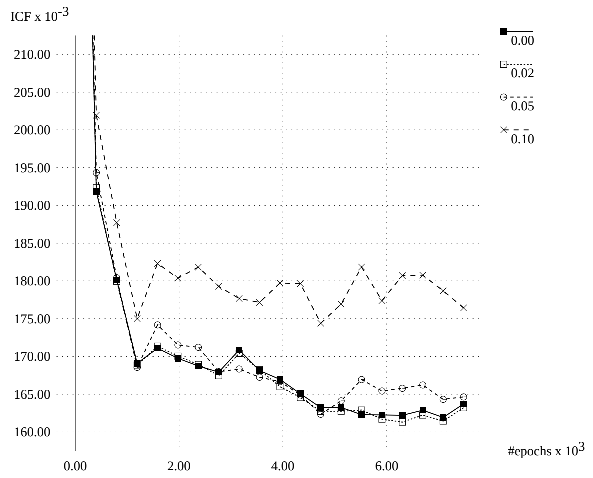


Figure 5.42: Performance on the AD-test set using noise.

Adding noise to the training pattern which is never more than 0.02 seems to improve the performance on the test set and will therefore be used when assessing the performance on the validation set.

Using weight decay

Incorporating the basic back-propagation algorithm with a weight decay term results in figures 5.43 and 5.44, presenting the results on the training and test set.

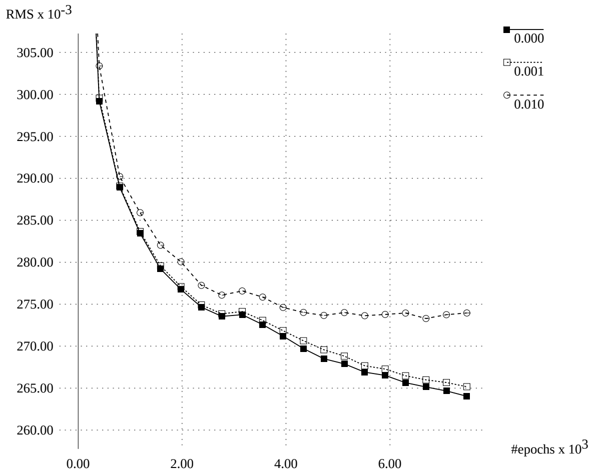


Figure 5.43: Performance on the AD-training set using weight decay.

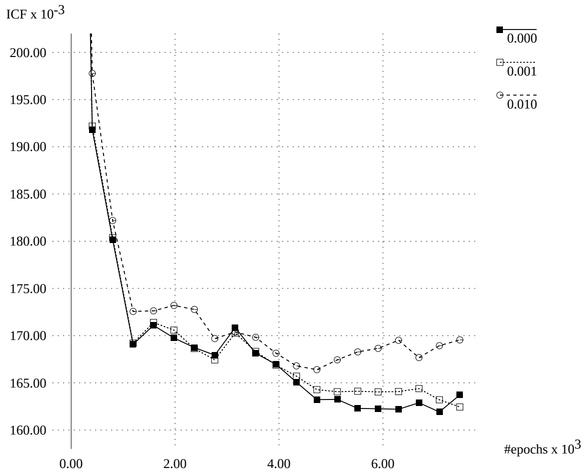


Figure 5.44: Performance on the AD-test set using weight decay.

The introduction of the weight decay term does not give a better performance and will therefore not be used in the remainder of this section.

Performance on the validation set

In figure 5.45 the performance on the validation set of the 5-40-2 network using a learning rate of 0.01 and a momentum term of 0.80 can be found. To each pattern of the training set noise of at most 0.02 has been added, but no weight decay has been used.

The network classifies best after 358 epochs; the classification percentages of this network are given in matrix 5.16.

$$\begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ \\ D \end{array} & \left(\begin{array}{cc} 86.0 & 14.0 \\ & \\ 15.4 & 84.6 \end{array} \right) \end{array} \qquad (5.16)$$

5.4.6 Discrimination between A and E

In this section the results of the discrimination between the classes A and E are presented. For a discussion on the results is referred to section 5.5.

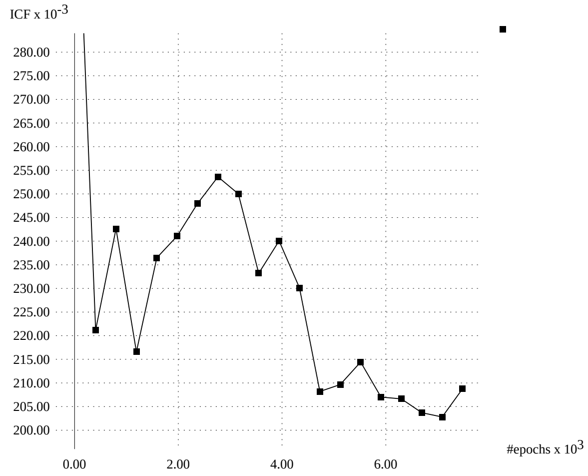


Figure 5.45: Performance on the AD-validation set for the optimal network.

Varying the architecture

The same variations in the network architecture as in the previous sections result in figures 5.46 to 5.47 for the training set, and figures 5.48 to 5.49 for the test set.

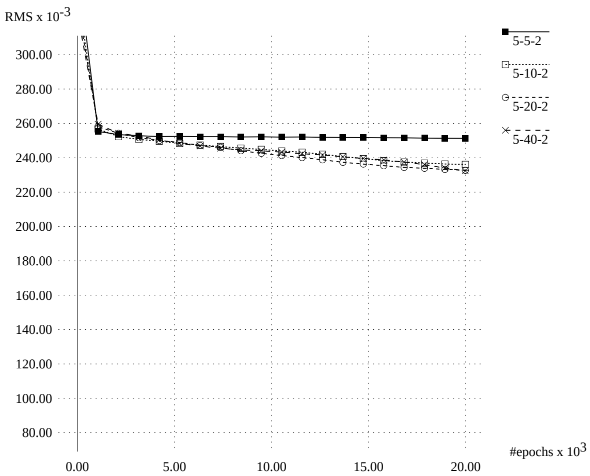


Figure 5.46: Performance on the AE-training set for several three-layer architectures.

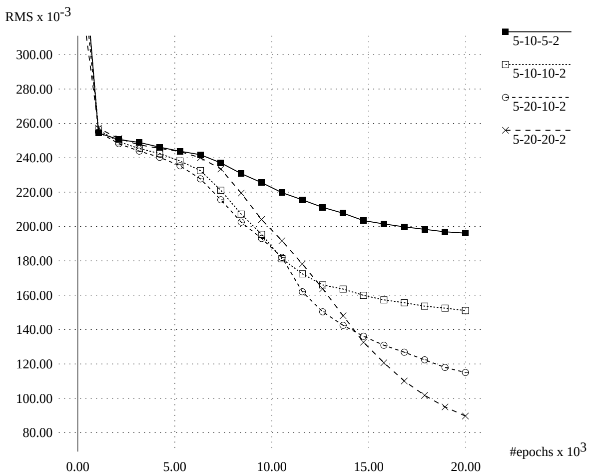


Figure 5.47: Performance on the AE-training set for several four-layer architectures.

It can be observed that most architectures achieve their best results on the test set in the first 7500 epochs, which is why in the remainder of this sections the number of epochs is limited to 7500. In this interval the architecture that gives the best performance is the one with one hidden layer containing 20 neurons.

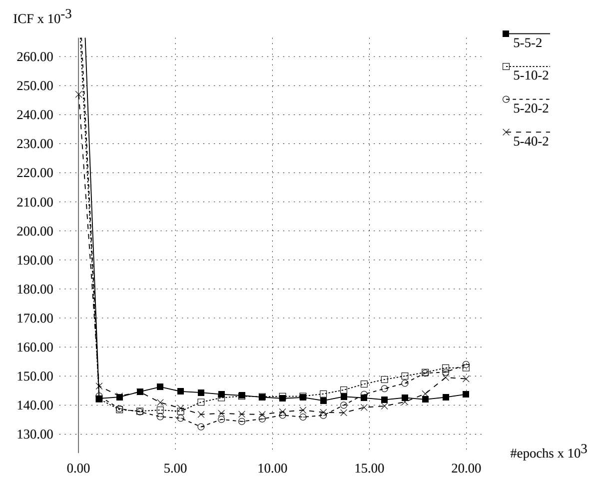


Figure 5.48: Performance on the AE-test set for several three-layer architectures.

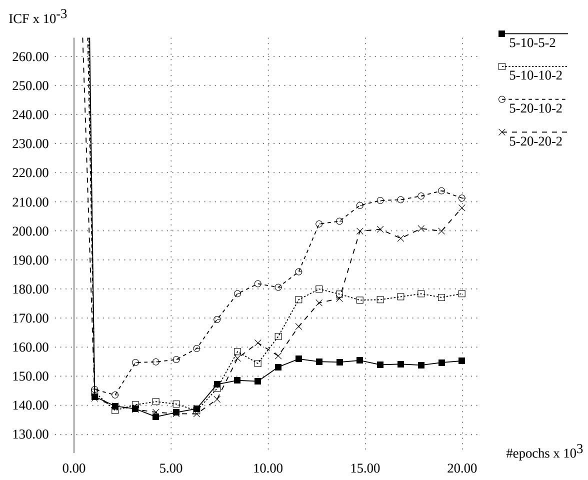


Figure 5.49: Performance on the AE-test set for several four-layer architectures.

Varying the learning rate

Trying learning rates of 0.2, 0.01 and 0.0005 gives figures 5.50 and 5.51, showing the results on the training respectively test set.

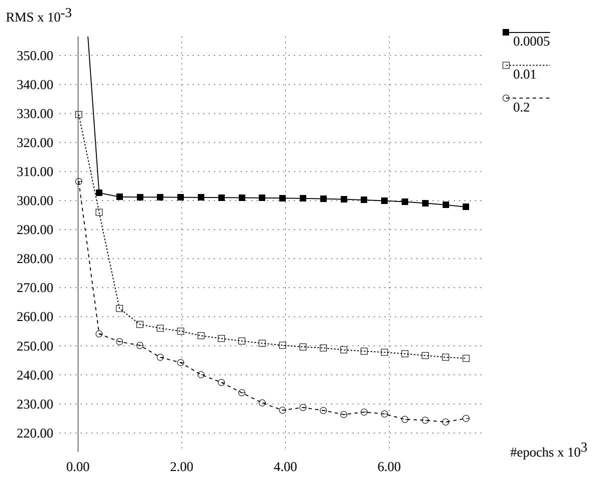


Figure 5.50: Performance on the AE-training set for several learning rates.

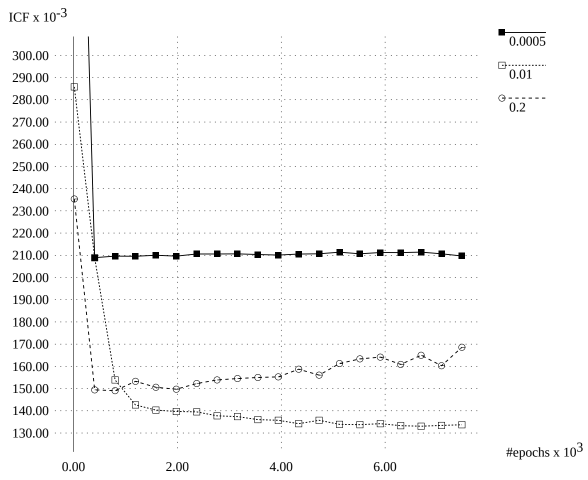


Figure 5.51: Performance on the AE-test set for several learning rates.

For this problem as well a learning rate of 0.01 is the best choice.

Varying the momentum term

If the magnitude of the momentum term is varied, the performance on the training set can be found in figure 5.52, while figure 5.53 displays the performance on the test set.

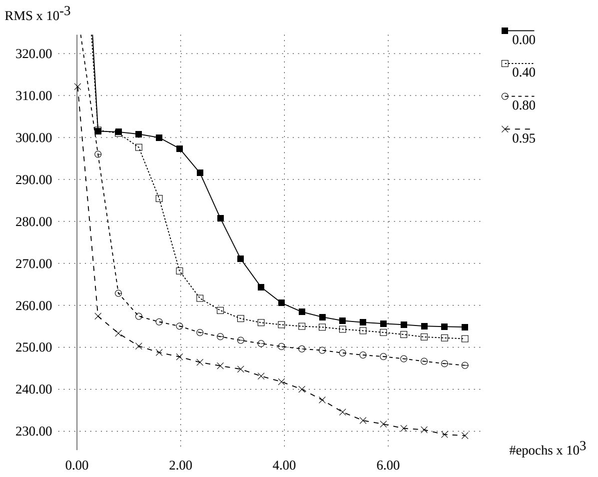


Figure 5.52: Performance on the AE-training set for several momenta.

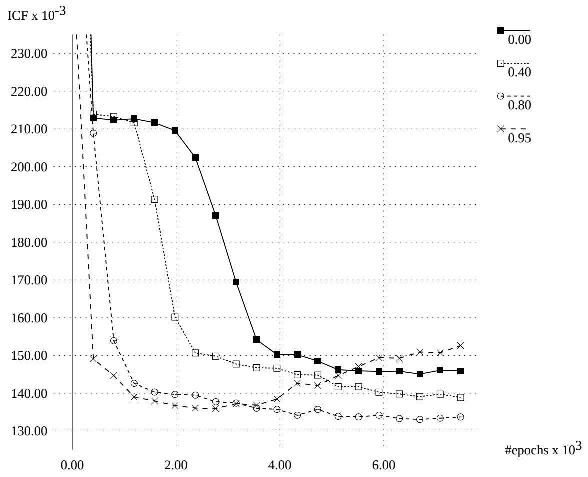


Figure 5.53: Performance on the AE-test set for several momenta.

The best performance on the test set is achieved with a momentum of 0.80.

Adding noise

In figures 5.54 to 5.55 the influence of adding noise to the training patterns can be found for the training and test set.

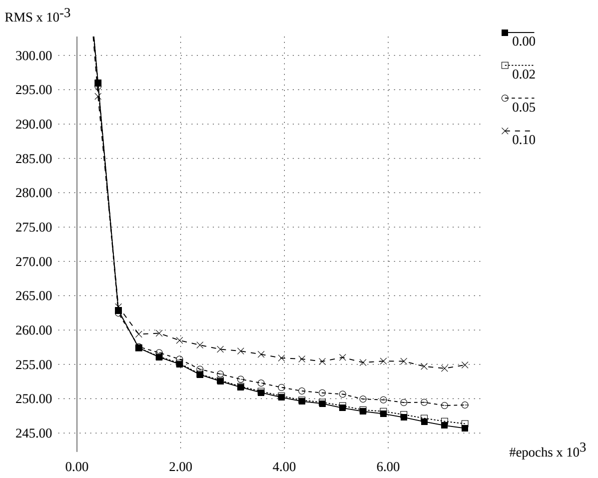


Figure 5.54: Performance on the AE-training set using noise.

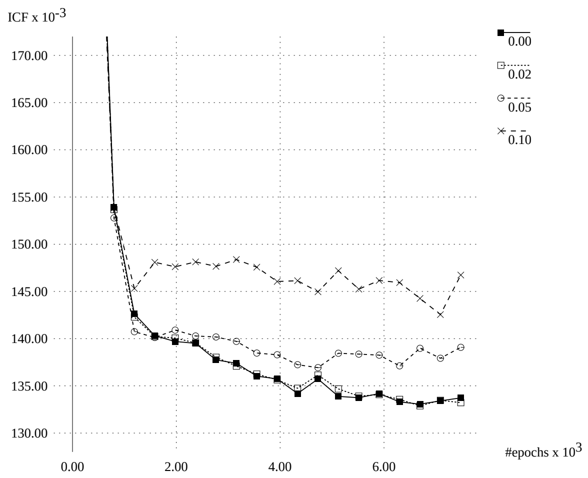


Figure 5.55: Performance on the AE-test set using noise.

The addition of noise during the training process gives no improvement of the classification results and is therefore not considered in the remainder of this section.

Using weight decay

The use of a weight decay term in the back-propagation algorithm results in figure 5.56 for the training set, and in figure 5.57 for the test set.

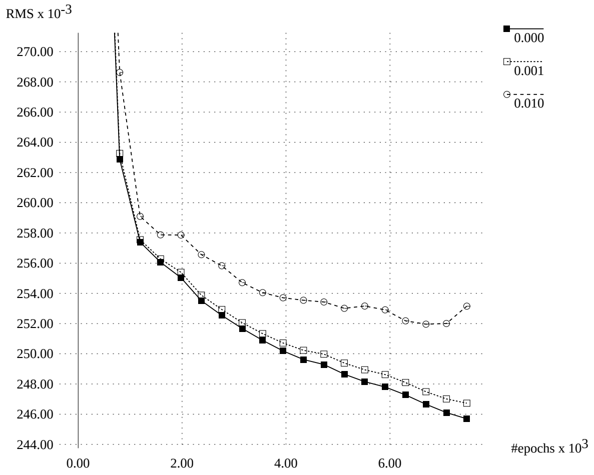


Figure 5.56: Performance on the AE-training set using weight decay.

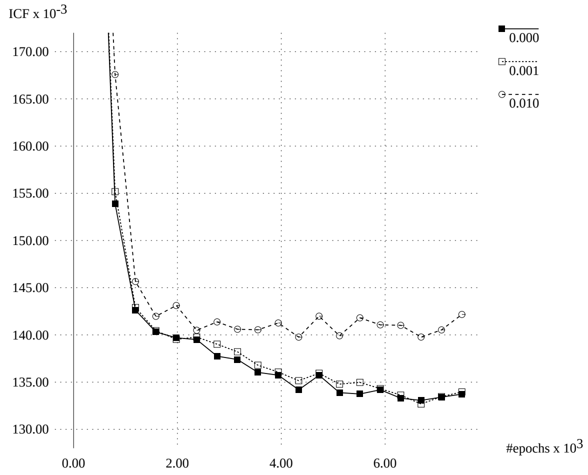


Figure 5.57: Performance on the AE-test set using weight decay.

The introduction of the weight decay term does not give a better performance and will therefore not be used when assessing the performance on the validation set.

Performance on the validation set

If the validation set is presented to the 5-20-2 network with a learning rate of 0.01, a momentum term of 0.80 and neither noise nor weight decay is used, the performance is as displayed in figure 5.58.

The best classification is reached after 2047 epochs, and is given in confusion matrix 5.17.

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 92.2 & 7.8 \\ & \\ 20.0 & 80.0 \end{pmatrix} \end{array} \qquad (5.17)$$

5.4.7 Discrimination between A, B, C, D and E

This section deals with the problem of discriminating between all five classes simultaneously. The results are discussed at a global level in section 5.5.

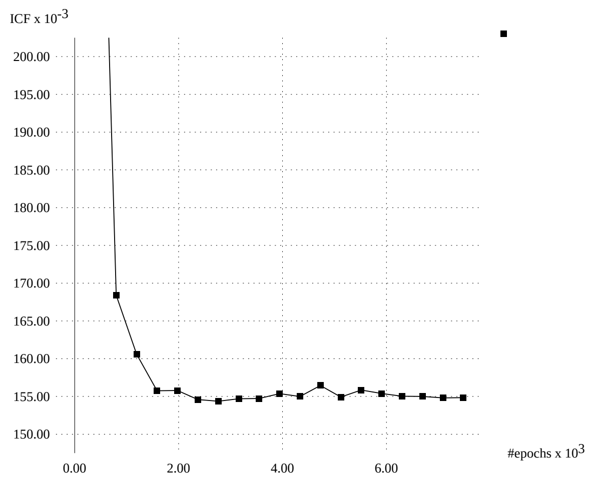


Figure 5.58: Performance on the AE-validation set for the optimal network.

Varying the architecture

The performance achieved by the examined architectures can be found in figures 5.59 to 5.60 for the training set, and in figures 5.61 to 5.62 for the test set.

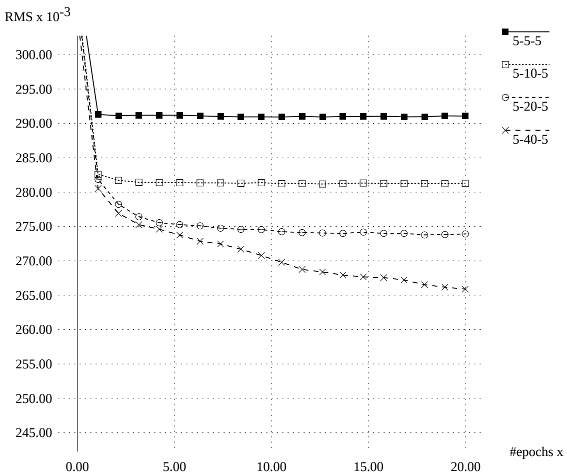


Figure 5.59: Performance on the ABCDE-training set for several three-layer architectures.

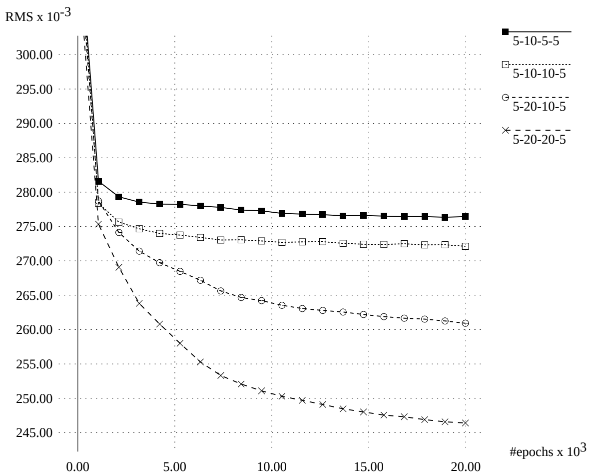


Figure 5.60: Performance on the ABCDE-training set for several four-layer architectures.

For most architectures the best performance on the test set is achieved during the first 7500 epochs, which is why in the remainder of this section the simulations comprise this number of epochs. In this interval the network with a single hidden layer containing 40 hidden neurons gives the best results.

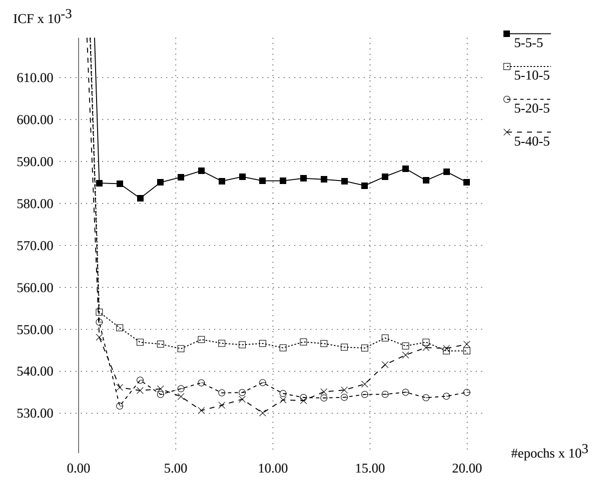


Figure 5.61: Performance on the ABCDE-test set for several three-layer architectures.

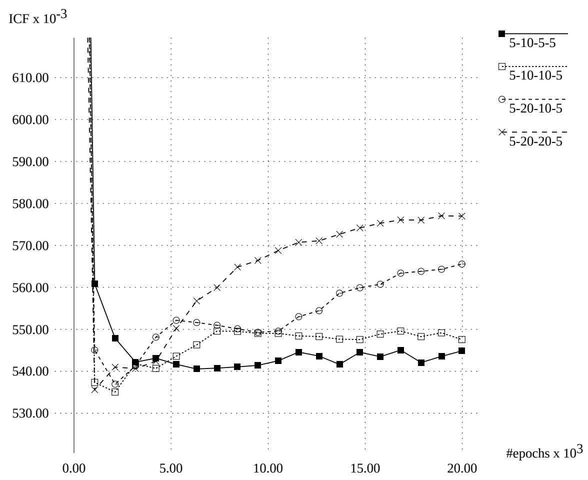


Figure 5.62: Performance on the ABCDE-test set for several four-layer architectures.

Varying the learning rate

In figure 5.63 the influence of variations in the learning rate on the performance on the training set is presented, while figure 5.64 shows the performance on the test set.

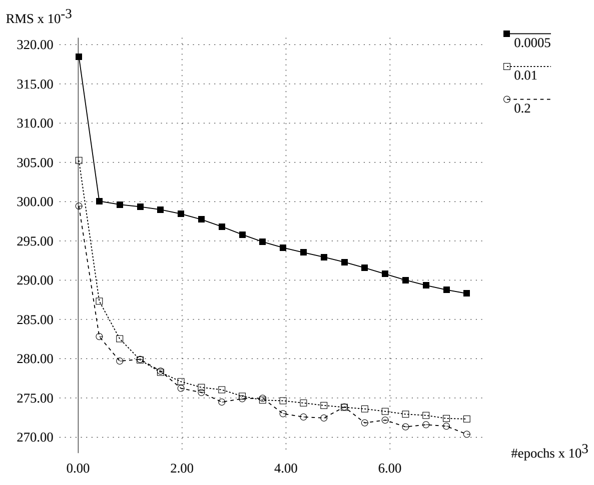


Figure 5.63: Performance on the ABCDE-training set for several learning rates.

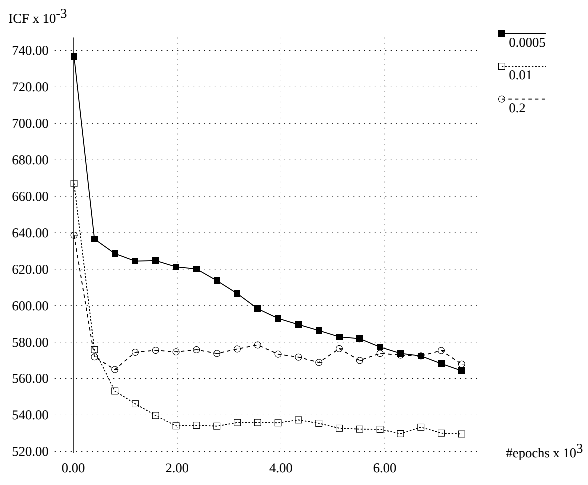


Figure 5.64: Performance on the ABCDE-test set for several learning rates.

The best choice for the learning rate seems to be 0.01.

Varying the momentum term

Simulations with different magnitudes of the momentum term result in figures 5.65 and 5.66, displaying the performance on the training respectively test set.

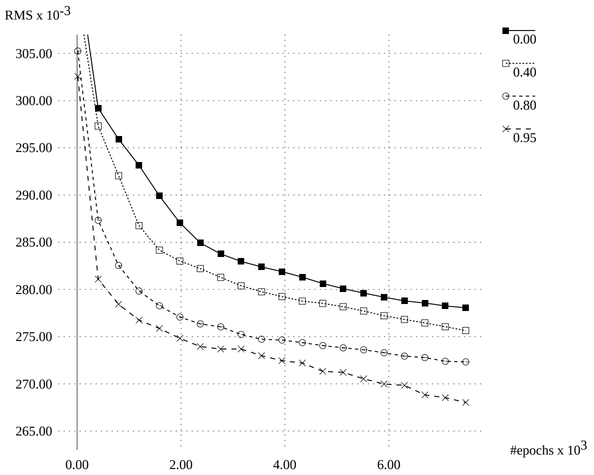


Figure 5.65: Performance on the ABCDE-training set for several momenta.

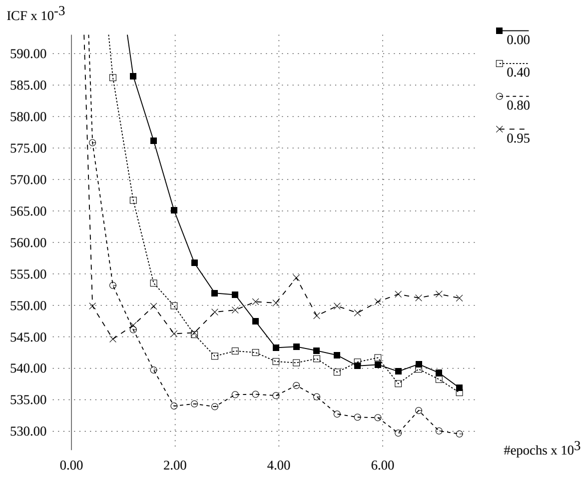


Figure 5.66: Performance on the ABCDE-test set for several momenta.

A momentum term of magnitude 0.80 gives the best results on the test set.

Adding noise

If during the training process the patterns of the training set are distorted by noise, the performance of the network changes to figure 5.67 for the training set, and to figure 5.68 for the test set.

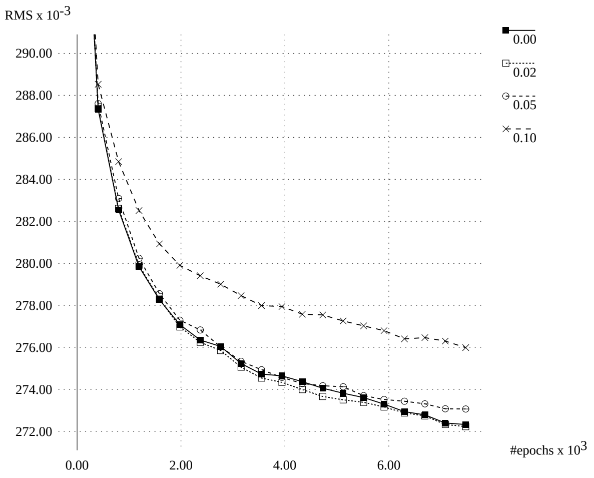


Figure 5.67: Performance on the ABCDE-training set using noise.

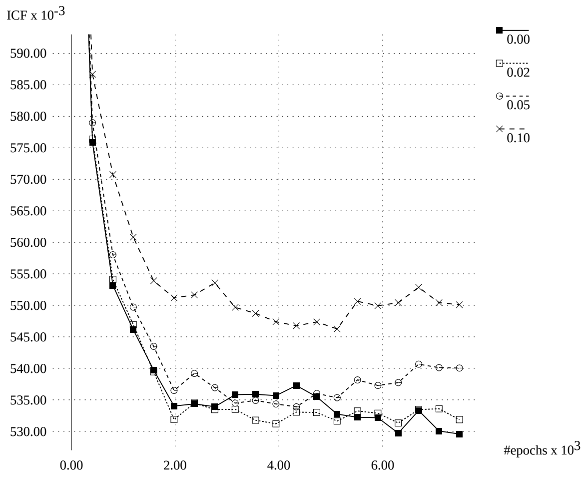


Figure 5.68: Performance on the ABCDE-test set using noise.

For this problem the addition of noise gives no clear improvement of the classification fractions and is therefore not considered in the remainder of this section.

Using weight decay

The extension of the basic back-propagation algorithm with a weight decay term results in figures 5.69 and 5.70 for the training and the test set.

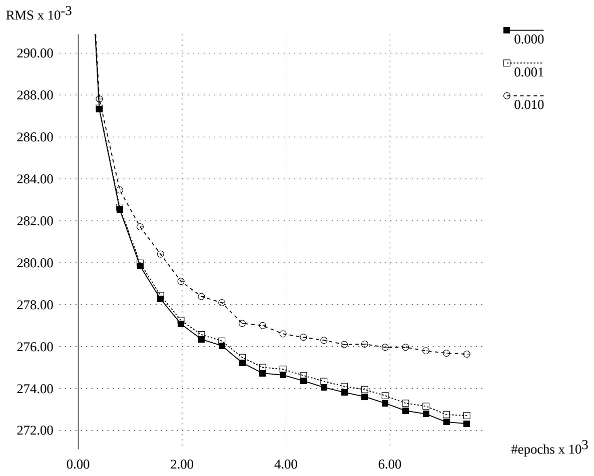


Figure 5.69: Performance on the ABCDE-training set using weight decay.

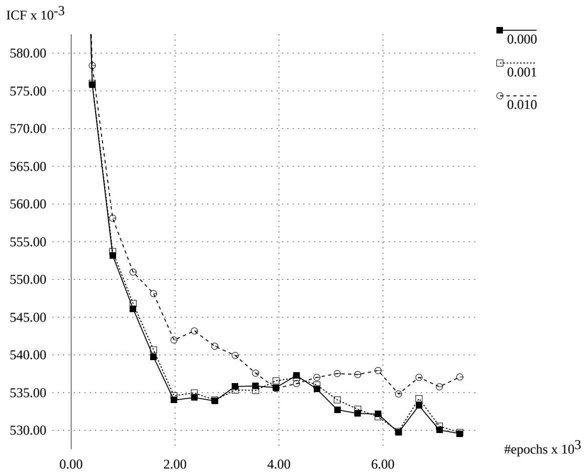


Figure 5.70: Performance on the ABCDE-test set using weight decay.

An additional weight decay term does not result in a better performance on the test set and will not be used for the determination of the performance on the validation set.

Performance on the validation set

The results on the validation set which are displayed in figure 5.71 are achieved by a 5-40-5 network with a learning rate of 0.01 and a momentum of 0.80, but without the application of either noise or weight decay.

The performance of this network is optimal after 3886 epochs; the classification percentages are given in matrix 5.18.

$$\begin{array}{cc} & \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{pmatrix} 80.6 & 0.0 & 12.4 & 3.1 & 3.9 \\ 0.0 & 80.0 & 0.0 & 10.0 & 10.0 \\ 14.3 & 14.3 & 64.3 & 7.1 & 0.0 \\ 15.4 & 7.7 & 15.4 & 46.2 & 15.4 \\ 20.0 & 10.0 & 10.0 & 0.0 & 60.0 \end{pmatrix} \end{array} \tag{5.18}$$

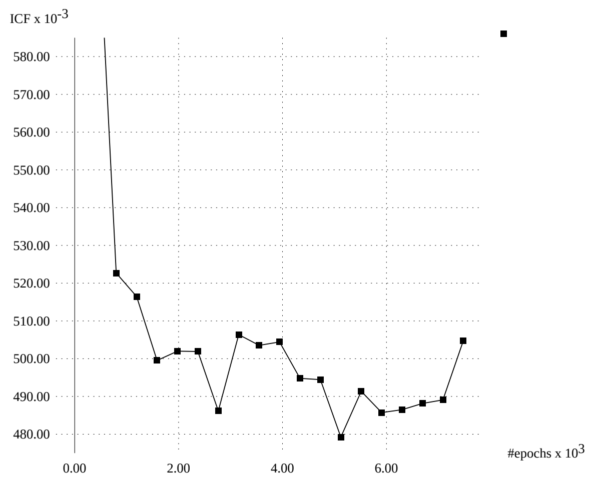


Figure 5.71: Performance on the ABCDE-validation set for the optimal network.

5.5 Discussion

In section 5.4 several parameters of a back-propagation network have been varied, resulting in a flood of figures. In this section the influence of these variations will be discussed, without specifically looking at one of the five problems.

Following the approach described in section 5.4.2 the architecture is the first aspect of the network that has been varied. In the according figures of the performance on the training set can be found that the larger the size of the network is, the better the training set is memorized, something which is in agreement with the theory. In contrast to the networks with only one hidden layer, of which the performance curves stay relative close to each other, the previous phenomenon applies much more to the networks with two hidden layers. Furthermore it can be observed that even the smallest four-layer network converges a lot better on the training set than the biggest three-layer network, although the number of hidden neurons is much smaller. When looking at the figures displaying the performance of the network on the test set, it are the networks with one hidden layer that achieve better results than the networks with two hidden layers. Combining the facts of a good convergence on the training set and a bad performance on the test set, it is obvious that the four-layer networks suffer a lot of over-learning (see section 5.3.3). We therefore conclude that for the problems that we have examined a network with a single hidden layer containing 20 to 40 neurons is the best choice.

The second parameter of the network that has been varied is the learning rate. In the figures displaying the RMS of the training set as a function of the number of epochs it can be found that the higher the learning rate, the better the convergence. It could be argued that the convergence on the training set for higher learning rates is not better but faster, but simulations for a few problems where the number of epochs had been enlarged to 50000 still showed a better convergence for a higher learning rate. If the performance on the test set is observed, a learning rate of 0.01 presents itself as the best choice; for a learning rate of 0.2 over-learning can again be detected.

For the momentum term more or less holds the same as for the learning rate: the higher the magnitude of the momentum term, the better the training set is memorized. However, a higher momentum term also entails a faster convergence, not only for the training set but also for the test set. For the best performance on the test set a momentum of 0.80 should be chosen, for all problems. Also for the momentum term over-learning can be observed, if α is set at 0.95.

The first variation on the basic algorithm, the addition of noise to the patterns of the training set, causes the network to memorize the training set less well as the amount of noise increases. This was to be expected, since the introduction of noise blurs the borders between the different classes, making a good discrimination between the classes even more difficult. Furthermore, since the training patterns continuously change during the training stage, the network gets no chance to tune itself optimally to the training set. Although the previously mentioned absence of tuning deteriorates the performance on the training set, it could improve the performance on the test set, because the network is prevented from over-learning. Unfortunately, in the figures displaying the performance on the test set no better results can be observed when noise is used. Usually the addition of noise results in a worse performance, especially if the amount of noise is relatively large, although in some cases — discrimination between classes A and D — a little noise can bring some improvement. Another negative aspect of adding noise is that the *ICF* of the test set starts to fluctuate as the noise gets stronger. The reason for this is that the network does not evolve slowly to some kind of optimum, but keeps changing because of the continuously changing patterns. Obviously this restlessness has its effect on the performance on the test set.

What in the previous paragraph was said about the influence of noise on the performance of the network largely holds for weight decay as well. The performance on the training set gets worse as the magnitude of the decay term increases, while the performance on the test set in some cases — A vs C — can improve a little, but usually deteriorates. An explanation for this behaviour is that, in order to solve the problem, the network needs weights with high values, while weight decay opposes to this. Therefore, as γ gets higher, the network gets more problems memorizing the training set as well as correctly classifying the test set.

When looking at the confusion matrices it can be seen that for all problems with two classes the percentages of correct classification are rather high, at least 80% but mostly higher. Although there is no explicit order in which the classes can best be discriminated, discrimination between class A and B can be performed extremely well, with classification percentages of respectively 93.8% and 100%. If there is to be discriminated between all five classes simultaneously, the percentages of correct classification naturally drop a bit for all classes, but for class D the deterioration is strikingly large. Probably class D is scattered much more over the observation space than the other classes, making a good discrimination of this class impossible. The order in which the classes can best be discriminated when using back-propagation on five classes is A/B, followed by C/E and concluded by D.

A final observation that can be made is that the performance curves of the test and the validation set in most cases differ a lot. Furthermore is the *ICF* of the validation set, except when discriminating between A and B, structurally higher than the *ICF* of the test set. The reason for this phenomenon is that the test set is generated in the same way as the training set, and therefore resembles the training set much more than the validation set. Since the network tries to achieve the best performance possible on the training set, a good performance on the test set is achieved automatically. The validation set on the other hand differs much more from the training set, resulting in a performance that is less well. Another reason for the previously mentioned differences in performance between the test and the validation set is that the number of patients in the validation set is much smaller. As a result of the small number of patients not only the percentage of correct classification (*ICF*) is low (high) when only a few patients of a small class are erroneously classified, but the *ICF* displays an unpredictable behaviour as well. All this however is not a flaw of the back-propagation algorithm, but simply the hard reality of a limited data set.

In the literature back-propagation has the reputation of being slow, something that has been confirmed by our simulations. Compared to the other methods of classification — see section 7.2 — back-propagation needed a lot of computation time to achieve good results. For the problem of discriminating between five classes simultaneously on the basis of 2500 patterns in the training set where a test set of another 2500 patterns was used to monitor the behaviour of the network, the 5-40-5 network needed for 7500 epochs about six days of computing on a Sun SPARCstation SLC. For the 5-20-20-5 network, which has 1.5 times the connections of the 5-40-5 network, the same workstation would have needed 24 days for 20000 epochs[2]!

[2]Fortunately for this simulation a Sun SPARCstation of the 600 MP series could be used; this computer completed the simulation within 10 days.

Chapter 6

Feature mapping

The method that is investigated in this chapter is feature mapping, an unsupervised method to train neural networks. How feature mapping works is explained in the first part of this chapter, while in the second part the results of the application of feature mapping to our data are presented. More on this subject can be found in [21, 22].

6.1 Network architecture

In 1982 Kohonen presented a new type of neural network called *self-organizing feature map*, which is very suitable for classification problems. The algorithm according to which the network is trained is unsupervised, meaning that the data does not have to be classified prior to presentation to the network.

A feature mapping network consists of two layers of neurons, an *input layer* and an *output layer*. All neurons in the output layer are fully connected to the neurons in the input layer, while between the neurons in both the input and the output layer no connections exist. The neurons in the output layer are usually organized in a two-dimensional grid, but a one-dimensional line or an arrangement in more than two dimensions is also possible. An example of a feature mapping network can be found in figure 6.1.

As will be explained in section 6.2.1 feature mapping is *topology preserving* or *topographic*, which means that pattern vectors that lie close to (far from) each other in the observation or *feature* space are mapped to neurons of the output layer that lie close to (far from) each other as well. Due to this non-trivial property feature mapping can be used to preprocess data: a data point is transformed to the position of the neuron in the output layer it is mapped to. Since the vector that describes the position of a neuron depends on the degree of organization of the output layer, the dimension of the data under observation can be changed to an arbitrary value; usually the dimension of the data is reduced.

Topographic maps also exist on the surface of the brain, for sensory and motor phenomena. The details of these maps however differ from self-organizing feature maps, not only in architecture — the maps of the brain have a much more complex architecture — but also in underlying principles.

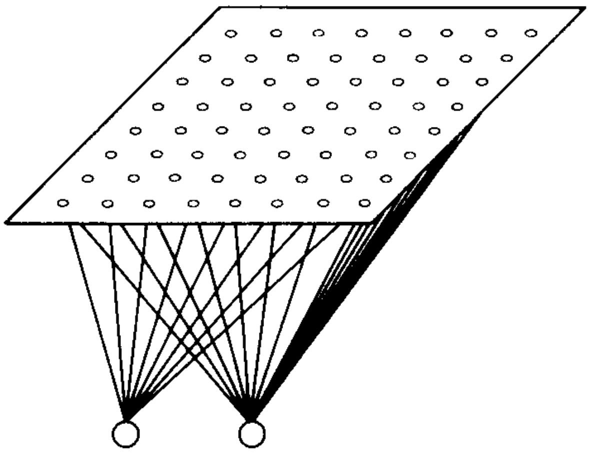


Figure 6.1: Network mapping a two-dimensional feature space onto a two-dimensional 8×8 grid.

6.2 Principles of the method

In the next sections the underlying principles of feature mapping are dealt with. Section 6.2.1 explains how a feature mapping network is trained and the topology preserving property is obtained. Since feature mapping is an unsupervised method, classification of data is not straightforward; sections 6.2.2 and 6.2.3 bring some clarity into this matter.

6.2.1 Learning

In section 6.1 we mentioned that a feature mapping network consists of only two layers that are mutually fully connected. If we see the incoming weights of a neuron of the output layer as a vector, every neuron of the output layer represents a position in the feature space. Training of a feature mapping network is now nothing more than positioning the neurons of the output layer in the feature space in a way that is optimal in some sense.

For a feature mapping network the error function that is to be minimized is given by

$$E(w) = \frac{1}{2} \sum_p \min_j \sum_i (v_{p,i} - w_{ji})^2 \tag{6.1}$$

with w representing the entire set of weights, w_{ji} the weight from neuron i to neuron j and $v_{p,i}$ the i-th element of patient vector p; p is an index over all patients, j over all neurons of the output layer and i over all neurons of the input layer.

In equation 6.1 it can be seen why scaling of the vectors or input patterns is necessary. If a particular element of the input pattern has a variance that is much larger than the variances of the other elements, it has a much bigger influence on the error function than the other elements, although the other elements could have a better discriminating ability. If now all elements are scaled to the same interval in the manner described in section 3.5, this disadvantage is taken away.

For the self-organizing feature map Kohonen has given the following learning rule for updating the weights with respect to the previous error function:

$$\Delta w_{ji} = \eta \sum_{p} \Lambda(j, j^*)(v_{p,i} - w_{ji}) \tag{6.2}$$

In this equation η represents the *learning rate*, which is a constant determining the speed of the training process, j^* is the index of the neuron closest to pattern p, and Λ stands for the *neighbourhood function*, described later in this section. Although equation 6.2 suggests that updating of the weights is only done after all patterns have been presented, this is not true. Instead of this approach, which is, similarly to back-propagation, called *learning by epoch*, often *learning by pattern* is chosen, where the weights are updated after presentation of every pattern. Learning by pattern creates stochastic noise that can be useful for avoiding local minima.

Feature mapping is a *competitive* learning method, which means that the neurons of the output layer compete to represent the currently presented pattern. The winner of this process is the neuron that has the smallest distance to the input pattern in the feature space. Usually the Euclidean distance is taken, so the winning neuron (index j^*) is defined by

$$\forall_j \ |w_{j^*} - v_p| \leq |w_j - v_p| \tag{6.3}$$

where j is an index over all neurons of the output layer, w_j stands for the set of incoming weights of neuron j and $|.|$ represents the Euclidean distance function.

In equation 6.2 appears a so called neighbourhood function Λ, defining a neighbourhood around the winning neuron j^*. If the neurons of the output layer are organized in grid, assigning to each neuron a position (x,y), the neighbourhood function is usually defined by

$$\begin{aligned} \Lambda(j, j^*) &= 1, \ \sqrt{(x_j - x_{j^*})^2 + (y_j - y_{j^*})^2} \leq d \\ &= 0, \ \text{otherwise.} \end{aligned} \tag{6.4}$$

In other words Λ is a function that indicates whether or not a neuron j is situated within the circular neighbourhood with radius d of neuron j^*. For a higher dimensional organization of the output layer extension of the neighbourhood function is straightforward.

It is the application of the neighbourhood function that accomplishes the earlier mentioned *topology preserving* property. This property embodies that pattern vectors that lie close to (far from) each other in the feature space are mapped to neurons of the output layer that lie close to (far from) each other as well. In the training stage the winning neuron and all neurons in its neighbourhood are moved towards the pattern that is to be learned, while the other neurons are left undisturbed. This results in an organization of the neurons of the output layer where an adjacent position in the output layer corresponds with an adjacent position in the feature space. The topology preserving property is now obvious, since the neuron to which a pattern is mapped depends on the position of that neuron in the feature space.

In an untrained feature map the neurons of the output layer usually have positions in the feature space that are randomly chosen. To create a topographic map the neurons of the output layer first have to be moved towards positions in the feature space roughly corresponding to their position in the output layer. For this purpose a large neighbourhood — read radius — and learning rate are needed. After this is achieved the goal of the training process should turn to tuning the positions of the neurons to get a good representation of the input patterns. For this purpose the neighbourhood and the learning rate must be small. A frequently used method to meet these requirements is linearly decreasing both the neighbourhood radius and the learning rate. The neighbourhood radius is now given by

$$d(t) = d_0 \left(1 - \frac{t}{T}\right) \tag{6.5}$$

and the learning rate by

$$\eta(t) = \eta_0 \left(1 - \frac{t}{T}\right) \tag{6.6}$$

with t representing the number of times the complete data set has already been presented to the network, the *number of epochs*, and T representing the total number of epochs the network is to be trained, so it always holds that $0 \leq d(t) \leq d_0$ and $0 \leq \eta(t) \leq \eta_0$.

An example of the topology preserving property, taken from [21], can be found in figures 6.2 to 6.4, in which the feature map of figure 6.1 is trained with a set of points uniformly drawn from $[0.0, 1.0] \times [0.0, 1.0]$. In these figures the positions of the neurons of the output layer in the two-dimensional feature space are displayed by a point, while their relative position in the 8×8 grid is displayed using lines. The initial weights of the network are randomly drawn for the interval $[0.45, 0.55]$, as can be seen in figure 6.2. After 6000 epochs — see figure 6.3 — the feature map already displays the topology preserving property, but does not fully occupy the feature space yet. In figure 6.4 the map is shown after training has been completed. The neurons at the edges of the grid are situated relatively close to their neighbours because these neurons are not pulled very hard to the border of occupied feature space, since outside the borders no patterns are situated.

In figure 6.5 the same map is shown after being trained with samples of which 42% is uniformly distributed over the shaded area, and 58% is distributed over the rest of the $[0.0, 1.0] \times [0.0, 1.0]$ area. The neurons of the output layer are again positioned in a way that reflects the positions of the input patterns over the feature space. Notice that for the patterns in the shaded area much finer differences can be discerned, since the density of the neurons is much higher in this area than in the rest of the feature space.

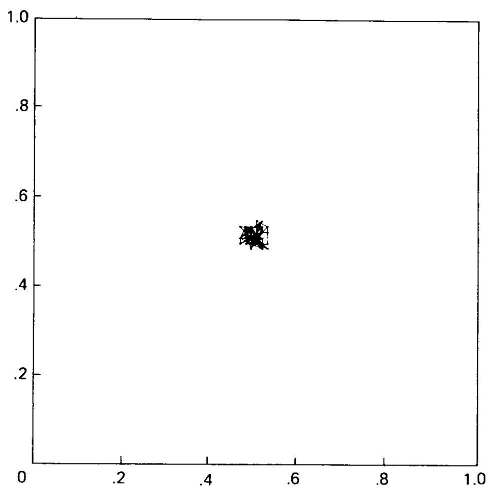


Figure 6.2: Positions of the neurons before training.

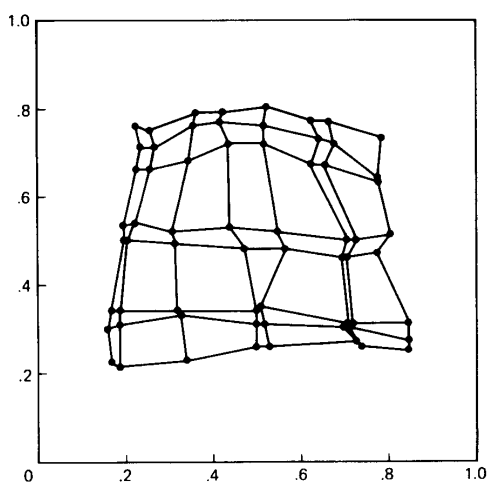


Figure 6.3: Positions of the neurons after 6000 epochs.

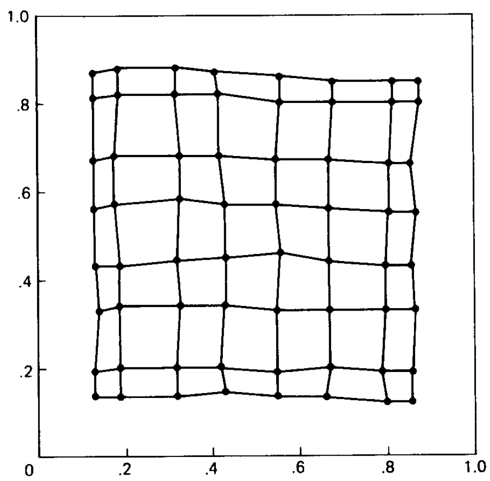


Figure 6.4: Positions of the neurons after 20000 epochs.

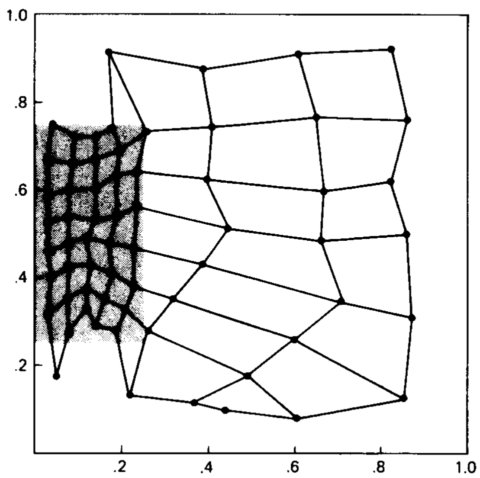


Figure 6.5: Positions of the neurons after 20000 epochs, when it is trained with non-uniform data.

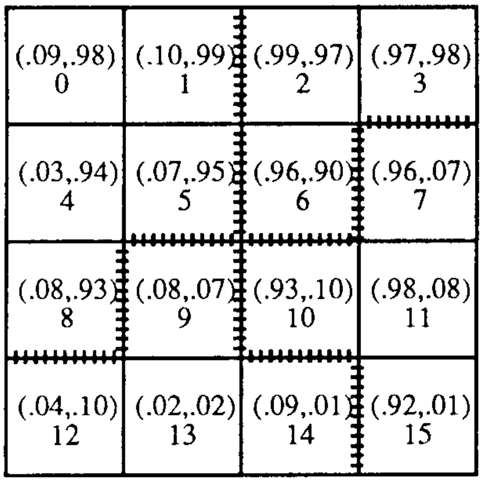


Figure 6.6: An two-dimensional map with the position of each neuron in the two-dimensional feature space in brackets.

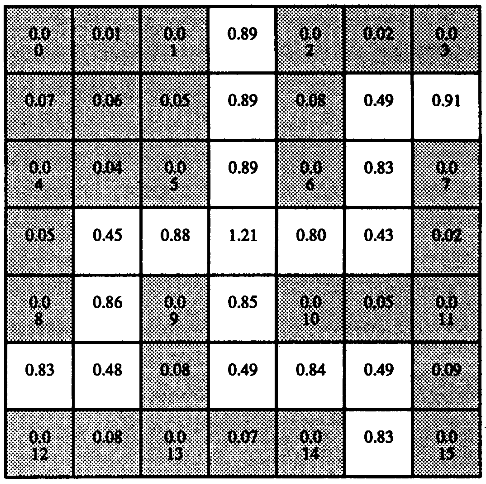


Figure 6.7: The U-matrix of the map in figure 6.6; the white entries represent the borders between the clusters when a water level of 0.4 is used.

6.2.2 Clustering

After the network has been trained, the map can be examined to detect *clusters*. A cluster consists of neurons that are neighbours on the map, and whose Euclidean distance to each other in the feature space is smaller than a certain value. These clusters have emerged because the data itself contains clusters and the feature map is topology preserving, something which has been explained in section 6.2.1. To determine the borders between the clusters a data structure called the *U-matrix* is used [24], containing information about the feature space distances between adjacent neurons on the map.

If a two-dimensional map with D^2 entries is used, the U-matrix will contain $(2D-1)^2$ entries. The entries in the U-matrix that have both an odd row as well as an odd column number, represent the neurons on the map. The entries that are positioned between two neurons that are horizontally or vertically adjacent on the map, have a value equal to the Euclidean distance between these neurons. The remaining entries, which lie on the intersection of two pairs of diagonally adjacent neurons, contain the average of the Euclidean distance of the two neuron pairs. An example of a feature map can be found in figure 6.6, while the corresponding U-matrix can be found in figure 6.7.

To explain the next step in the clustering process a different view on the U-matrix is taken. The U-matrix is regarded as an altitude map of some landscape, meaning that the entries of the matrix are interpreted as altitudes; the entries representing neurons are assigned a zero altitude. Now the landscape is flooded, until the water reaches a certain level selected by the user. This will cause the entries with values smaller than the water level to disappear under water. Neurons whose corresponding U-matrix entries lie in the same puddle are considered to belong to the same cluster. It is obvious that the quality of the clustering process depends very much on the selection of the water level. If the level is chosen too low, many insignificant clusters will appear; if on the other hand the level is chosen too high, clusters that are significantly different will be joined.

6.2.3 Classification

Kohonen's self-organizing feature map is an unsupervised neural network, meaning that the data does not have to be classified prior to presentation to the network. Classification of a vector in this case can entail no more than determining the cluster to which the winning neuron belongs. Although sometimes this is all there is to be accomplished, this is usually not the case.

For most applications a classification based on the problem domain is asked for. In our case for example there are, depending on the problem, two or five classes to be distinguished. A solution to this difficulty is to create a translation from the clusters to the classes of the problem domain. To find this translation some expert knowledge is necessary. One possibility is to take some vectors out of every cluster and find out what class they really belong to some way or the other, hoping that the chosen vectors are representative for their cluster.

However, if the data on the basis of which the feature map is constructed is supervised, the translation can be found much easier. The only thing that has to be done for each cluster is keeping track of the classes to which the vectors belong that have a neuron in that cluster as winner: the cluster is labeled with that class most vectors belong to that a have winning neuron in the cluster.

As mentioned in section 6.2.2 it is very important to choose the right water level, for a too high level will create less clusters than actual classes, thus making classification of vectors to some classes impossible. A level that is too low will create a great number of small clusters, some of which can not be labeled because not a single vector is placed in it.

6.3 Variations on the basic algorithm

In the course of years many variations on the basic algorithm for feature mapping have been proposed. The most drastic change was introduced by Kohonen himself, converting his algorithm from an unsupervised into a supervised one, called *Learning Vector Quantization* or LVQ. Others have suggested changes in definition of the neighbourhood function, the function that determines the winning neuron etc. In section 6.3.1 such a variation is discussed.

6.3.1 Conscience

The function of Kohonen for determining the winning neuron, the minimal Euclidean distance, is very simple and usually gives a good network performance. Sometimes however this function can cause a relatively small group of neurons being the winner relatively often, while many other neurons remain unused. The unused neurons do not have any influence on the network performance, but do slow down the training process. In 1988 Desieno [22, 24] therefore proposed a variation in the basic algorithm to cancel this disadvantage.

Desieno provided every neuron of the output layer with a *conscience*, making it for frequently winning neurons difficult to win again. The conscience is achieved by the following bias term

$$b_j = \gamma \left(\frac{1}{n\_outputs} - c_j(t) \right) \tag{6.7}$$

where j is an index over all neurons of the output layer, index t is increased every time a pattern is presented to the network, $n\_outputs$ equals the number of neurons of the output layer and γ is a constant that is usually set at 10. The term $c_j(t)$ actually implements the conscience and is defined by

$$\begin{aligned} c_j(0) &= 1 \\ c_j(t+1) &= (1+\beta)c_j(t), \text{ if neuron } j \text{ is the winner} \\ &= (1-\beta)c_j(t), \text{ otherwise} \end{aligned} \tag{6.8}$$

with β being a constant usually set at 0.0001. Equation 6.3, according to which the winning neuron (index j^*) is determined, now changes to

$$\forall_j \ |w_{j^*} - v_p| - b_{j^*} \leq |w_j - v_p| - b_j \tag{6.9}$$

where j is an index over all neurons of the output layer, w_j represents the set of incoming weights of neuron j, v_p is the pattern currently presented to the network and $|.|$ represents the Euclidean distance function.

It is easy to understand that due to the conscience mechanism it is impossible that a neuron never becomes a winner. For by repeatedly being a looser neuron, the neurons conscience keeps improving in comparison to the other neurons consciences. Finally the neuron must be declared the winner, independent of its position in the feature space relative to the position of the input vectors. Although it is not simple, by a good choice of γ and β a network consisting of n neurons in the output layer can be forced to declare each neuron on average $\frac{1}{n}$-th of time to be the winner.

6.4 Results

The results which are presented in the remainder of this chapter are obtained with the Kohonen neural networks simulator by Bierman. A short users-manual of this simulator can be found in [24], while our application specific alterations of the simulator — *koho* — are described in appendix A.

6.4.1 Measuring the performance

The way in which the performance of a feature map is measured is more or less equal to measuring the performance of a back-propagation network, something that is done deliberately to make a good comparison between the two methods possible. In this section therefore only the differences between back-propagation and feature mapping are dealt with, while for the common principles is referred to section 5.4.1.

Analogous to section 5.4.1 two error functions are used, the *RMS* and the *ICF*. The *RMS*, which is derived from the internal error function described in equation 6.1, is given by

$$RMS(w) = \sqrt{\frac{\sum_p \min_j \sum_i (v_{p,i} - w_{ji})^2}{n\_patients\ n\_inputs}} \tag{6.10}$$

Just like in equation 6.1 w represents the entire set of weights, w_{ji} the weight from neuron i to neuron j and $v_{p,i}$ the i-th element of patient vector p; p is an index over all patients, j over all neurons of the output layer and i over all neurons of the input layer. In addition $n\_patients$ stands for the number of patients and n_inputs for the number of neurons in the input layer. This error function is no longer dependent on the magnitude of the data set, nor of the number of features that is used to characterize a patient.

The fraction of incorrect classification that we are interested in is given by the *ICF*, defined by

$$ICF(w) = \sqrt{\frac{\sum_c \left(\frac{n\_patients_c - \sum_p \Psi(c_p, l_{p^*})}{n\_patients_c} \right)^2}{n\_classes}} \tag{6.11}$$

In this equation c is an index over all classes, $n\_classes$ is the number of classes, p is an index over all patients in the class under observation, and $n\_patients_c$ is the number of patients in class c. The symbol Ψ stands for a function that returns 1 when the class patient p actually belongs to (c_p) is equal to the class label (l_{p^*}) that was assigned to the cluster containing the winning neuron, and 0 otherwise. It is obvious that the *ICF* can only be defined if the data is supervised.

Although the *ICF* is defined in equation 6.11 using a function Ψ, while in the back-propagation equivalent 5.13 a different function Φ is used, both variants are comparable. The difference is caused by the differing principles of the methods: for back-propagation incorrect classification of a patient has to be checked by comparing the target pattern with the output pattern of the network, while for feature mapping the class of a patient has to be compared with the label that has been assigned to the cluster of the winning neuron. Since both Φ as well as Ψ return 1 in case of correct classification and 0 otherwise, both definitions of the *ICF* give results that are completely equal.

Just like in case of back-propagation, for a feature mapping network the *RMS* and *ICF* are not fully correlated, as can be seen in figure 6.8. This is not surprising, since the feature mapping network has no knowledge of the *ICF* either. Again the *ICF* is a bit jumpy; in the case of feature mapping this can be explained by the way in which it is determined what the label of a class is. Suppose that classes X and Y overlap in such a way that some set of vectors of class X as well an approximately equally large set of class Y are mapped

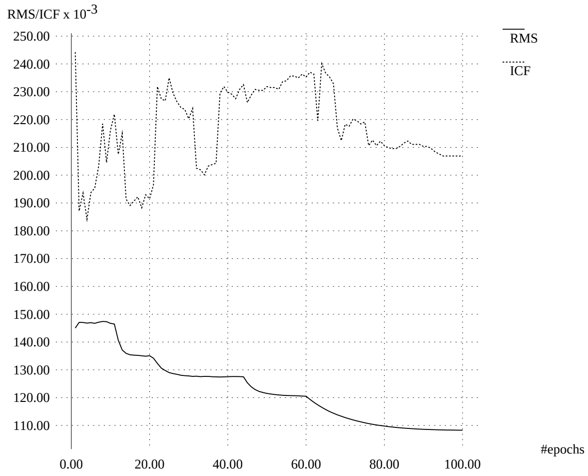


Figure 6.8: Comparison between the *RMS* and *ICF* for a feature mapping network.

to a certain cluster, and the cluster is labeled with class X. The cluster only has to be translated a little towards class Y to change the label of the cluster from X to Y, resulting in a decreased performance for class X and an increased performance for class Y.

6.4.2 Approach to solving the problem

For feature mapping we take the same approach to find the optimal network as for back-propagation, which was described in section 5.4.2. In this section most aspects of this approach are summarized, while for a more extensive explanation is referred to section 5.4.2.

Just like with the other methods, we first try to discriminate between healthy people and every disease separately, before trying to discriminate between five classes simultaneously. The experience gained with the partial problems is used to get a feeling for the problem domain and to reduce the amount of work that has to be done to find the optimal network for the other problems.

A feature mapping network also has several parameters that have to be adjusted correctly to get a good performance. Again we try to determine their optimal values by varying them in order of decreasing expected influence. For the parameters whose optimal value have not been determined yet values are assumed based on rules of thumb and previous experience. After the optimal network using the basic algorithm has been found, we try Desieno's variation — adding a conscience to every neuron — to see if the performance improves.

During the training stage the performance of the network is monitored using the *RMS* for the training set and the *ICF* for the test set. Determining the best value for a parameter is mainly done on the ground of the *ICF*, while the *RMS* is only used when the *ICF* does not clearly favour a particular value. In all figures five successive points are averaged, not only to reduce the number of points, but also to reduce potential jumpy behaviour of especially the *ICF*.

Once a promising network has been found, the validation set is presented to it. From the figure displaying the *ICF* against the number of epochs the number of epochs is determined after which the network classifies the validation set best. The network in this stage is used to draw up a confusion matrix which is used for comparison with the other classification methods.

In the remainder of this section the initial adjustments of the parameters of a feature mapping network are briefly discussed. Only the architecture and the learning rate — in that order — are varied, while for the other parameters their adjustment is motivated.

Architecture

The architecture of a feature mapping network is the parameter that has the biggest influence on the performance and is therefore varied first, so no initial value has to be chosen.

The only aspect of the architecture that can be varied is the size of the output layer, the map itself. The size of the input layer depends on the dimension of the feature space and thus can not be varied. With relation to the connections nothing can be varied either, because in a feature mapping network no connections can exist between neurons in the same layer, while the neurons in the input layer must be fully connected to the neurons in output layer.

Just like with back-propagation it is possible to create a network that classifies the training set perfectly, the only requirement is that the network is large enough. Because in this case the network has probably overfitted the training set, the performance on the test set will probably be very poor.

Learning rate

Also for feature mapping networks the rule of thumb exists that the learning rate (η_0) should approximately be equal to 0.25 divided by the number of patterns in the training set when learning by pattern. However, it is our experience with the same data that easily a larger value can be chosen without the risk of heavy oscillation. To limit the number of epochs the learning rate is therefore initially adjusted to 0.005. During the training process it is automatically decreased according to equation 6.6.

Water level

As mentioned in section 6.2.3 the selection of the water level is very important to the process of clustering and thus classification. Because we want to avoid the situation that the number of clusters is less than the number of actual classes, thus making correct classification of one or more classes impossible, we adjust the water level to zero. This way for every neuron a separate cluster is created that in its turn is assigned a class label, making it unlikely that each class is not represented by at least one cluster as long as there is no class that is relatively small compared to the other classes. Creating this many clusters however is a little bit dangerous: it is possible that in some clusters not a single vector from the training set is placed, so to that cluster no label is assigned. For the performance on the training set this is not a problem, but it is possible that vectors from the test and validation set are placed in such a cluster, making classification impossible. It is our experience however that for our data this situation does not occur if the feature map is relatively small, and only in a limited number of cases if the feature map is larger.

Neighbourhood radius

The size of the neighbourhood is a parameter that does not have to be varied. It is common practice to adjust the initial neighbourhood radius (d_0) to a half or a third of the width of the feature map [21, p. 168]; we have chosen for half the width. During the training process the neighbourhood radius is automatically decreased using equation 6.5.

Initial weights

If one uses the basic feature mapping algorithm, there exist basically two methods for the initialization of the weights. The first, *random initialization*, places the neurons of the output layer on a random position within the smallest hypercube[1] containing all vectors in the training set. The second method, *standard deviation initialization*, first determines the mean and standard deviation of the training vectors, and then places the neurons randomly in the hypersphere[2] with the center at the mean position and a radius equal to the standard deviation.

The method that gives the best results is the one where the neurons are positioned relatively close to most of the training vectors, because this way the translations of the neurons can be limited. In this light random initialization can give a bad performance if a few training vectors are isolated from the rest, resulting in a much bigger hypercube than necessary. Many neurons can be placed on a position far from every vector, never becoming a winner and thus not contributing to a better classification, although slowing down the training process. With standard deviation however it is possible that the hypersphere contains only a fraction of the training vectors, because for each class the vectors are highly clustered while the centers of these clusters have relatively large distance to each other. Now some neurons will be translated towards the clusters, but the others will remain unused, only slowing down the training process. Since our data is generated in such a way that it is does not contain isolated vectors[3], we have chosen to use random initialization.

Learning grain

Just like with back-propagation, updating of the weights can take place after presentation of a single pattern — learning by pattern — or all patterns — learning by epoch — of the training set. In [22, p. 222] can be found that

[1] Of course, a three-dimensional feature space implies a cube, two-dimensions a rectangle and one-dimension a line segment.

[2] Lower dimensions imply successively sphere, circle and line segment.

[3] This property is acquired by the way in which we have scaled the data, see section 3.5.

learning by pattern combined with a random order in which the patterns are presented to the network is useful for avoiding local minima, so only this fine learn grain is considered.

6.4.3 Discrimination between A and B

This section deals with the discrimination between classes A and B. A description of the training, test and validation set can be found in section 3.5, while the results are discussed in section 6.5.

Varying the architecture

Eight different sizes for the feature map are tried, varying from 3x3 to 10x10. The performance of the maps on the training set can be found in figures 6.9 to 6.10, while the figures 6.11 to 6.12 show the results of the test set.

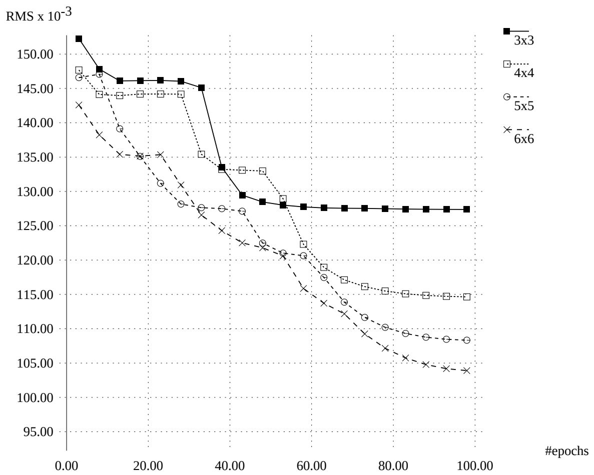


Figure 6.9: Performance on the AB-training set for several architectures.

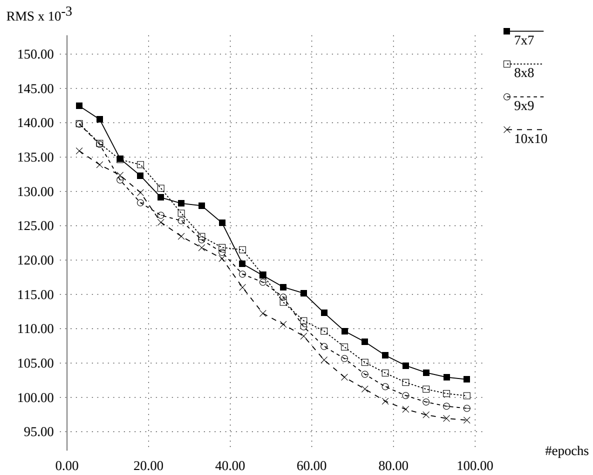


Figure 6.10: Performance on the AB-training set for several architectures.

We have chosen the 7x7 map for further optimizing.

Varying the learning rate

For the learning rate three values are tried: 0.1, 0.005 and 0.00025. The results on the training set are shown in figure 6.13, and the performance on the test set is presented in figure 6.14.

A learning rate of 0.005 seems to give the best performance.

Adding conscience

The results of adding conscience to each neuron as described in section 6.3.1 can be found in figure 6.15 for the training set, and in figure 6.16 for the test set.

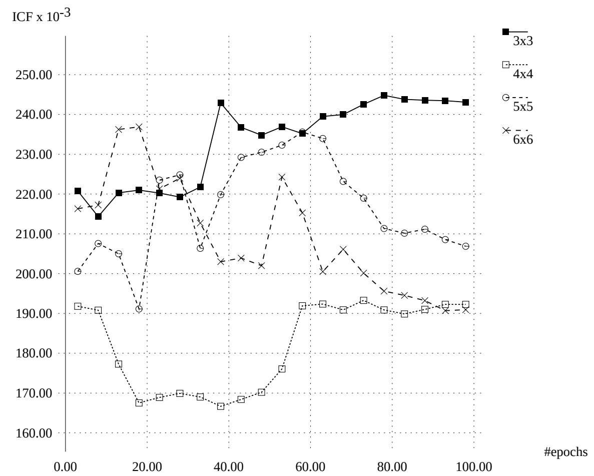


Figure 6.11: Performance on the AB-test set for several architectures.

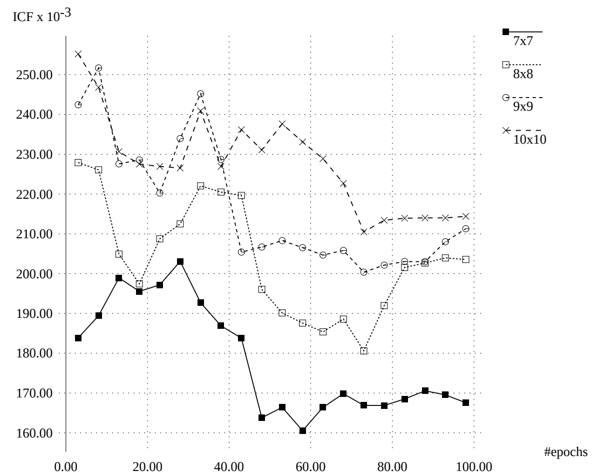


Figure 6.12: Performance on the AB-test set for several architectures.

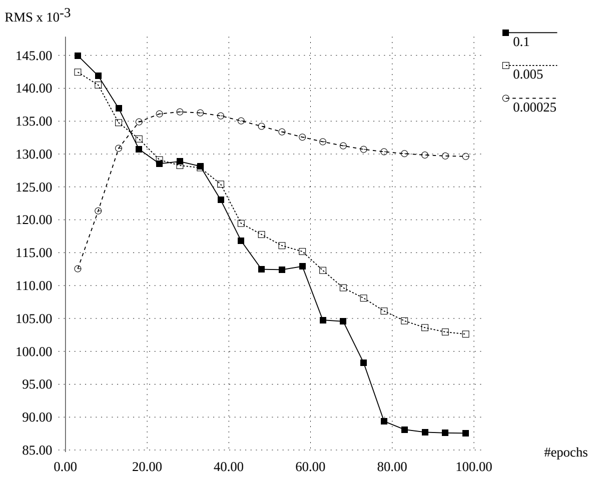


Figure 6.13: Performance on the AB-training set for several learning rates.

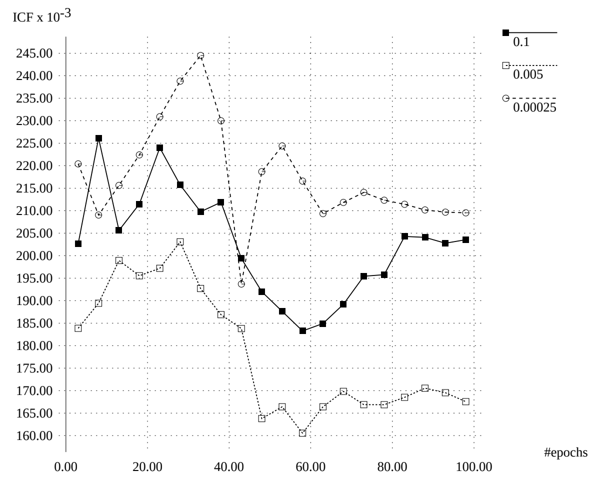


Figure 6.14: Performance on the AB-test set for several learning rates.

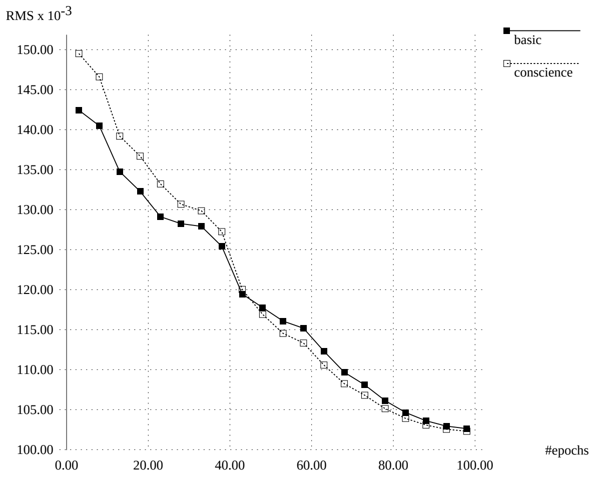


Figure 6.15: Performance on the AB-training set using conscience.

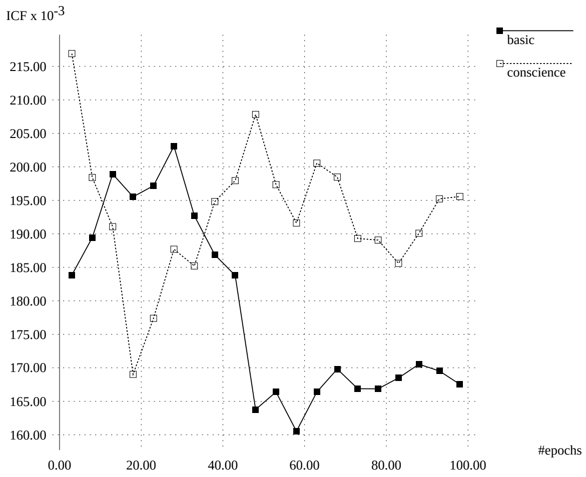


Figure 6.16: Performance on the AB-test set using conscience.

This variation on the basic algorithm does not improve the performance of the network and is therefore left out of consideration.

Performance on the validation set

The performance on the validation set of the 7x7 feature map with a learning rate of 0.005 without conscience is presented in figure 6.17.

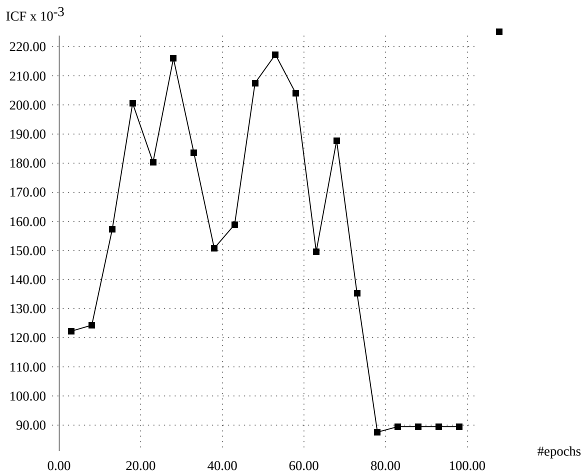


Figure 6.17: Performance on the AB-validation set for the optimal feature map.

The network gives the best performance after 73 epochs; the classification percentages of this situation are given in confusion matrix 6.12.

$$\begin{array}{ccc} & A & B \\ A & \multicolumn{2}{c}{\multirow{2}{*}{}} \end{array}$$

$$\begin{array}{c c} & \begin{array}{cc} A & B \end{array} \\ \begin{array}{c} A \\ \\ B \end{array} & \begin{pmatrix} 94.6 & 5.4 \\ \\ 10.0 & 90.0 \end{pmatrix} \end{array} \qquad (6.12)$$

6.4.4 Discrimination between A and C

This section deals with the problem of discriminating between classes A and C using feature maps. A discussion of the results can be found in section 6.5.

Varying the architecture

The same architectures as in section 6.4.3 are examined; the results on the training set are presented in figures 6.18 to 6.19, while the figures 6.20 to 6.21 show the results on the test set.

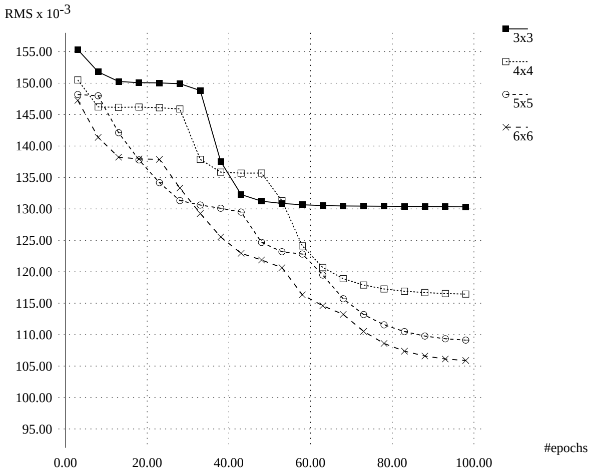


Figure 6.18: Performance on the AC-training set for several architectures.

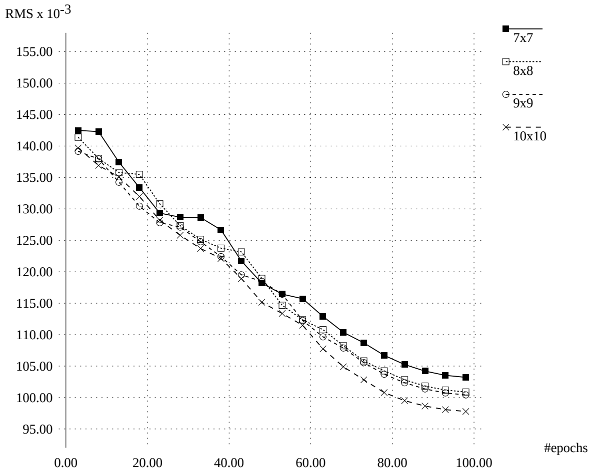


Figure 6.19: Performance on the AC-training set for several architectures.

The 4x4 map is the most promising, and is therefore used for further optimizing.

Varying the learning rate

The results of 4x4 feature maps with different learning rates can be found in figure 6.22 for the training set and in figure 6.23 for the test set.

The best performance is achieved with a learning rate of 0.005.

Adding conscience

Adding conscience to the neurons of the output layer gives the results shown in figures 6.24 and 6.25 for the training respectively test set.

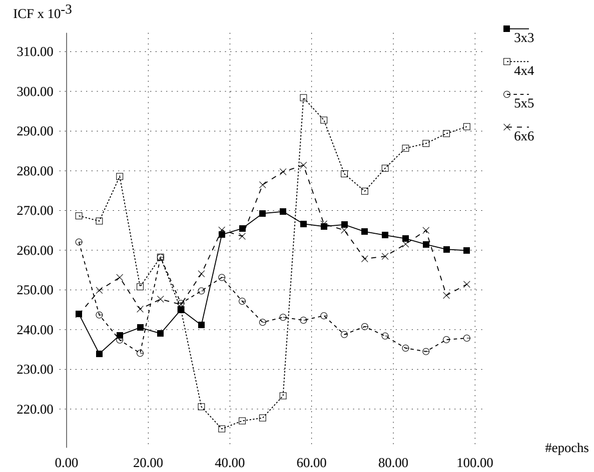


Figure 6.20: Performance on the AC-test set for several architectures.

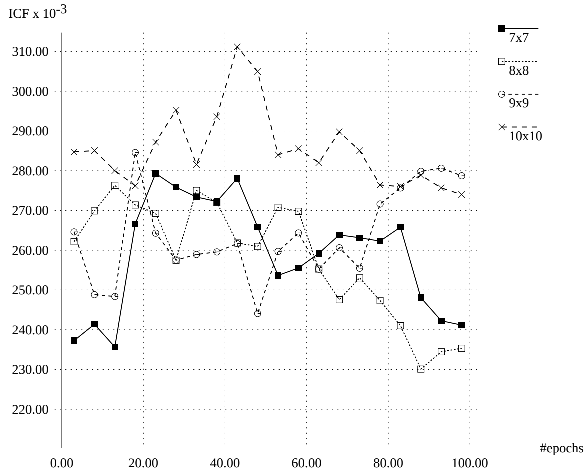


Figure 6.21: Performance on the AC-test set for several architectures.

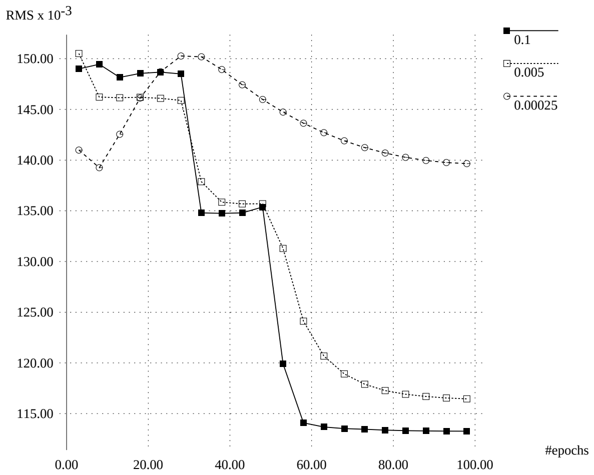


Figure 6.22: Performance on the AC-training set for several learning rates.

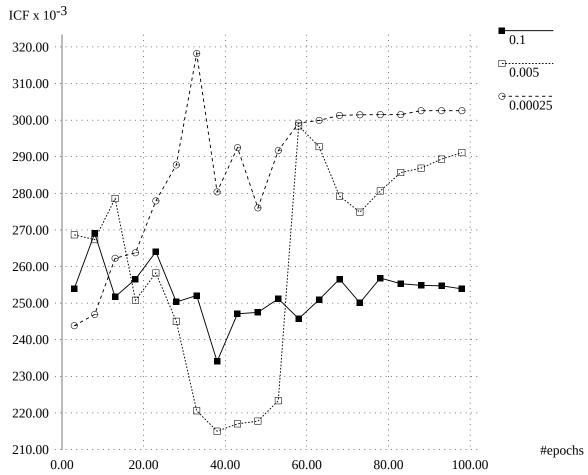


Figure 6.23: Performance on the AC-test set for several learning rates.

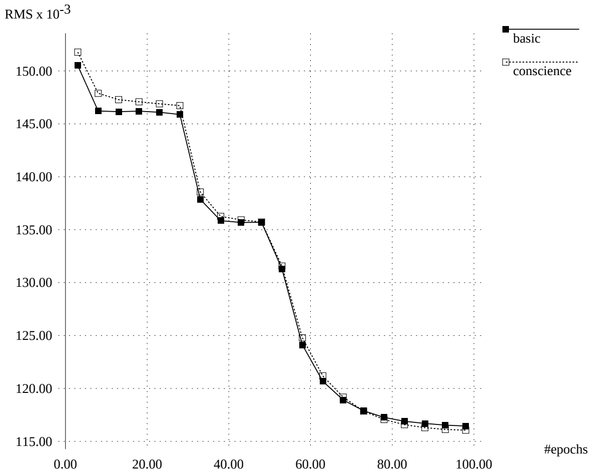


Figure 6.24: Performance on the AC-training set using conscience.

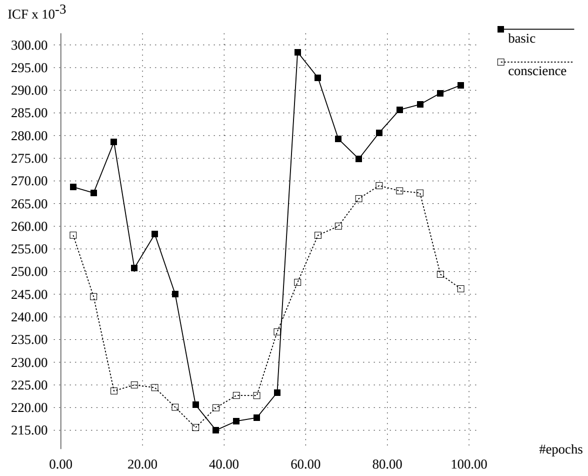


Figure 6.25: Performance on the AC-test set using conscience.

The algorithm that makes use of neurons with conscience seems to give a classification that is a fraction better than the basic algorithm does, and is therefore used in the remainder of this section.

Performance on the validation set

The 4x4 feature map with a learning rate of 0.005 and neurons with conscience gives a performance on the validation set as presented in figure 6.26.

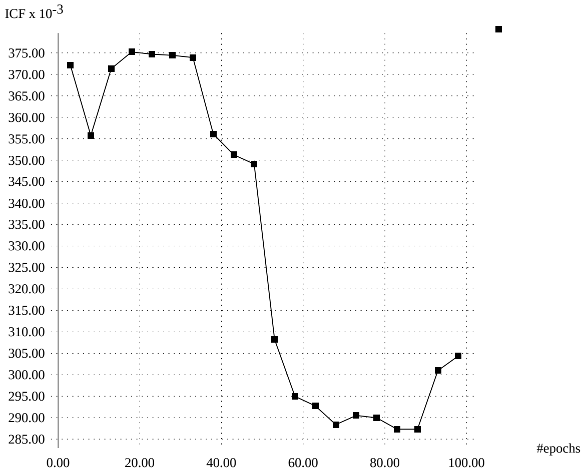


Figure 6.26: Performance on the AC-validation set for the optimal feature map.

The confusion matrix of the network after 68 epochs is given in matrix 6.13.

$$\begin{array}{ccc} & A & C \\ A & 80.6 & 19.4 \\ C & 35.7 & 64.3 \end{array} \tag{6.13}$$

6.4.5 Discrimination between A and D

The results of using feature mapping to discriminate between class A and D are given in this section. A discussion of these results can be found in section 6.5.

Varying the architecture

The performance of maps of size 3x3 to 10x10 can be found in figures 6.27 to 6.30. Figures 6.27 and 6.28 show the performance on the training set, while in figures 6.29 and 6.30 the performance on the test set can be found.

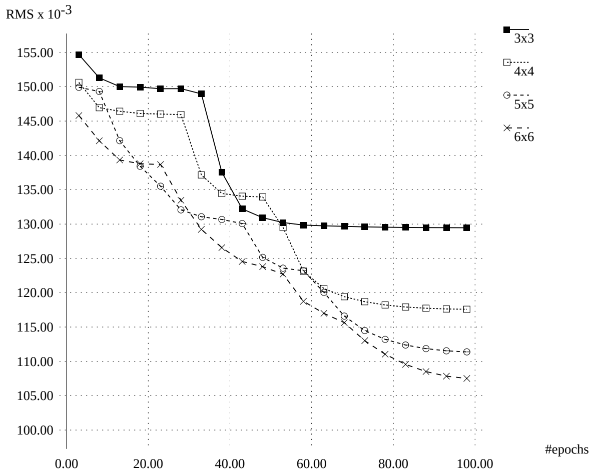


Figure 6.27: Performance on the AD-training set for several architectures.

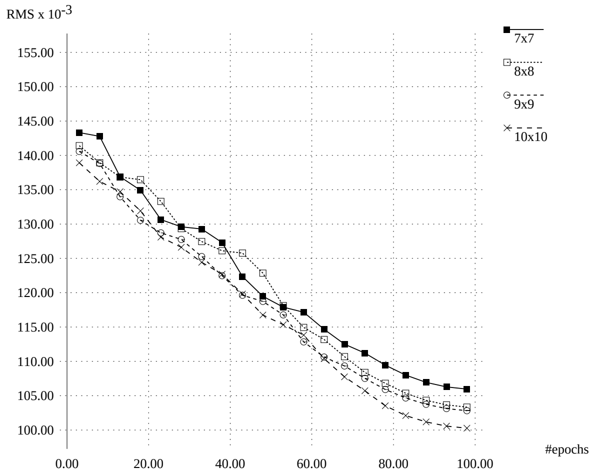


Figure 6.28: Performance on the AD-training set for several architectures.

The map with 8x8 output neurons gives the best results and is chosen as the default map for this problem.

Varying the learning rate

Variations in the learning rate result in figure 6.31 displaying the performance on the training set, and in figure 6.32 showing the performance on the test set.

A learning rate of 0.005 seems to be the best choice.

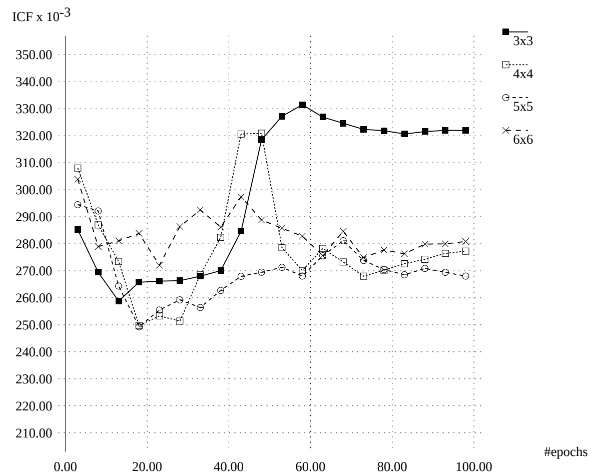


Figure 6.29: Performance on the AD-test set for several architectures.

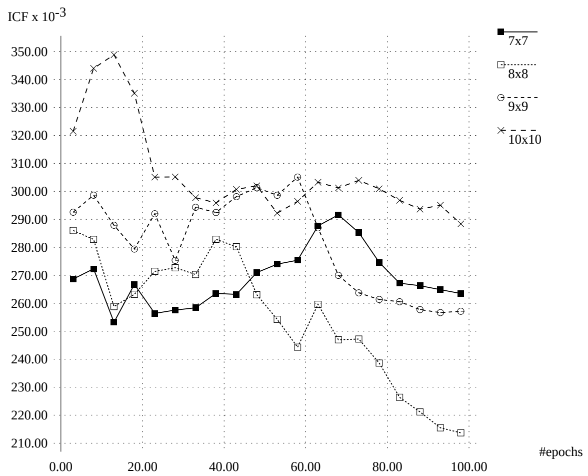


Figure 6.30: Performance on the AD-test set for several architectures.

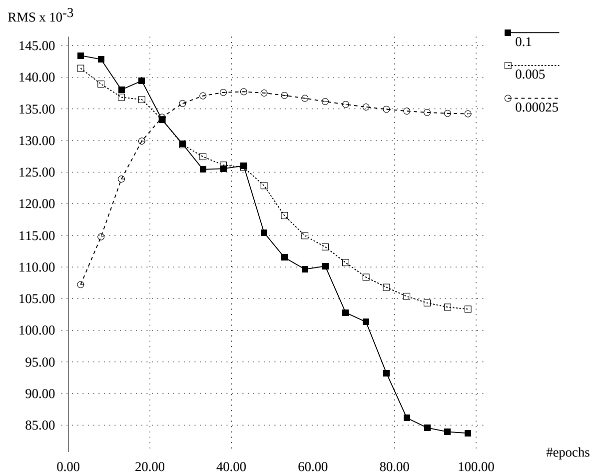


Figure 6.31: Performance on the AD-training set for several learning rates.

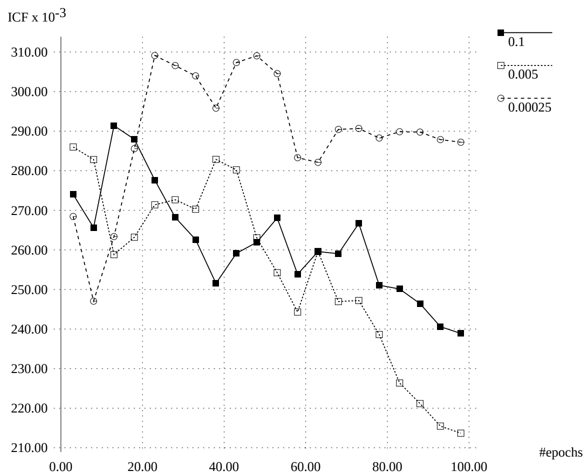


Figure 6.32: Performance on the AD-test set for several learning rates.

Adding conscience

The influence of providing each neuron with a conscience is shown in figure 6.33 for the training set, and in figure 6.34 for the test set.

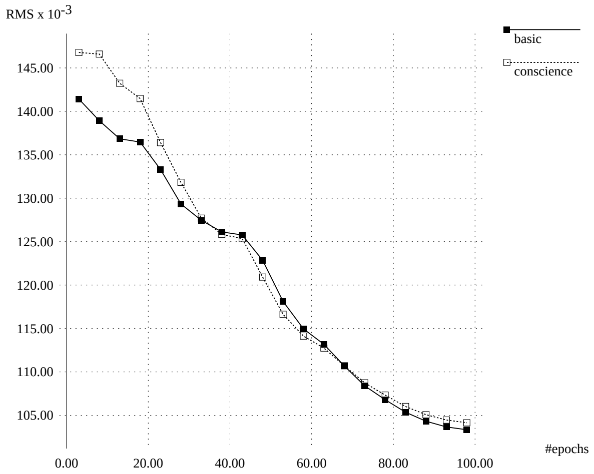


Figure 6.33: Performance on the AD-training set using conscience.

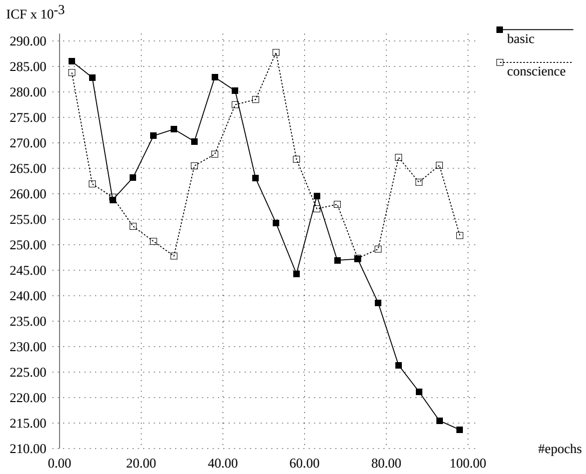


Figure 6.34: Performance on the AD-test set using conscience.

Since a conscience does not improve the performance on the test set, this variation is no longer considered.

Performance on the validation set

Figure 6.35 presents the performance of the 8x8 feature map with a learning rate of 0.005 and neurons without conscience on the validation set.

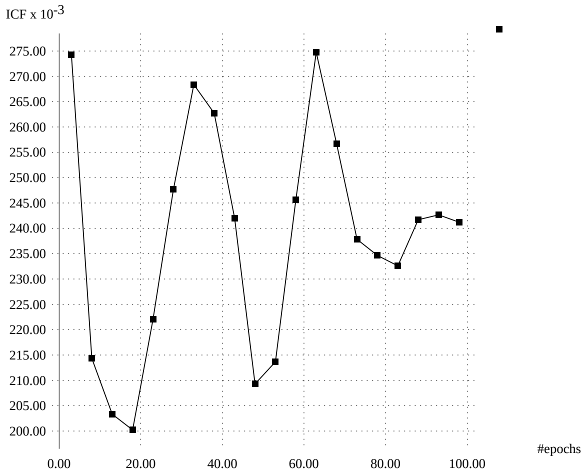


Figure 6.35: Performance on the AD-validation set for the optimal feature map.

The network classifies best after 16 epochs; the classification percentages of this network are given in matrix 6.14.

$$\begin{array}{cc} & \begin{array}{cc} A & D \end{array} \\ \begin{array}{c} A \\ \\ D \end{array} & \begin{pmatrix} 81.4 & 18.6 \\ & \\ 15.4 & 84.6 \end{pmatrix} \end{array} \tag{6.14}$$

6.4.6 Discrimination between A and E

In this section the results of the discrimination between the classes A and E are presented. For a discussion on the results is referred to section 6.5.

Varying the architecture

The well-known variations in the architecture give the results in figures 6.36 to 6.37 for the training set, and figures 6.38 to 6.39 for the test set.

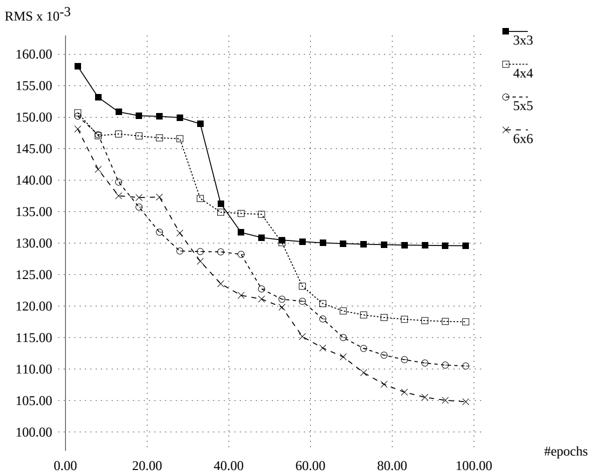


Figure 6.36: Performance on the AE-training set for several architectures.

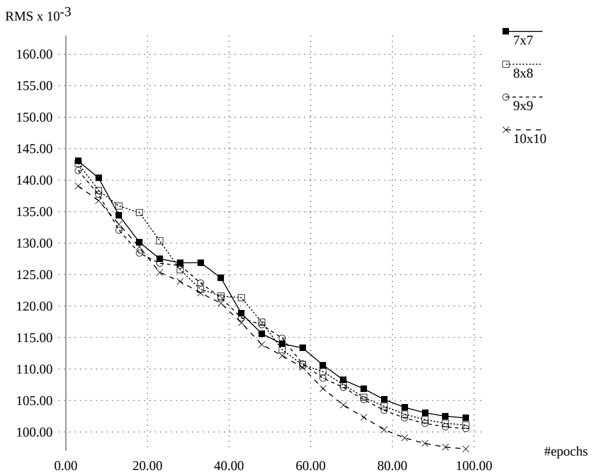


Figure 6.37: Performance on the AE-training set for several architectures.

Because with the 7x7 map the best performance is achieved, this map is used for further optimizing.

Varying the learning rate

Learning rates of 0.1 and 0.00025 besides the default value of 0.005 result in a performance on the training set as presented in figure 6.40, and a performance on the test set shown in figure 6.41.

For this problem a learning rate of 0.1 gives the best performance.

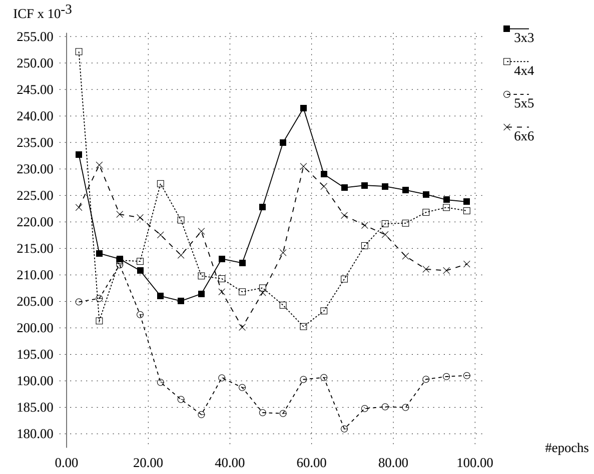


Figure 6.38: Performance on the AE-test set for several architectures.

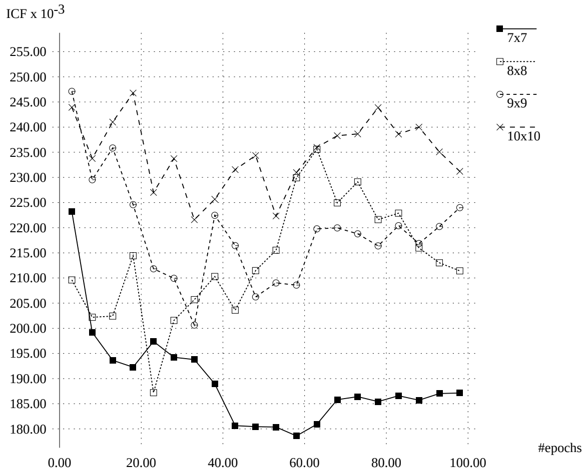


Figure 6.39: Performance on the AE-test set for several architectures.

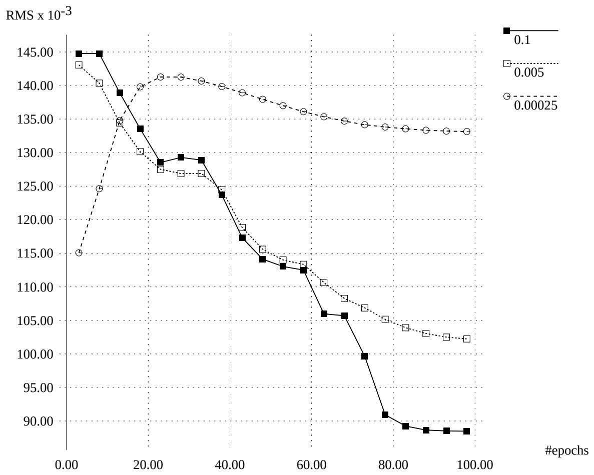


Figure 6.40: Performance on the AE-training set for several learning rates.

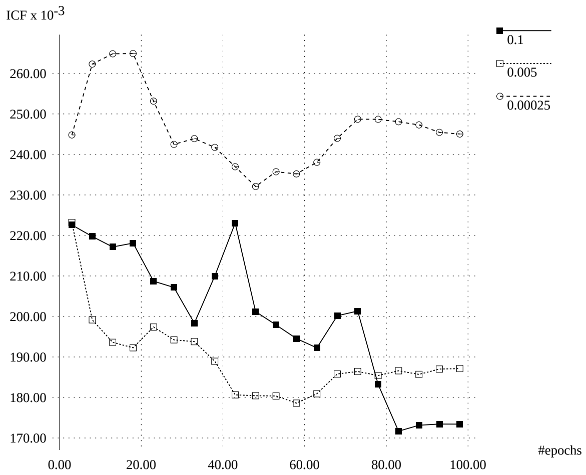


Figure 6.41: Performance on the AE-test set for several learning rates.

Adding conscience

In figure 6.42 the performance of the network with neurons with and without a conscience on the training set can be found, while in figure 6.43 the according performance on the test set is given.

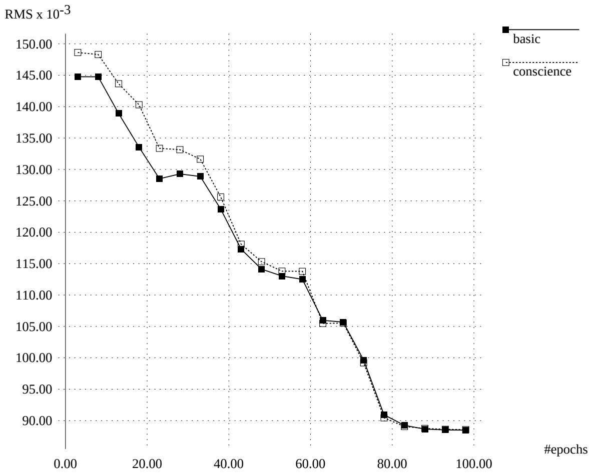


Figure 6.42: Performance on the AE-training set using conscience.

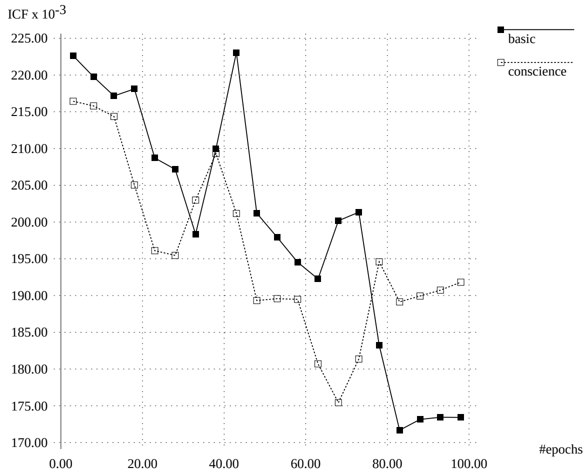


Figure 6.43: Performance on the AE-test set using conscience.

Again the basic algorithm gives a slightly better performance and is therefore used in the remainder of this section.

Performance on the validation set

If the validation set is presented to the 7x7 network with a learning rate of 0.1 and neurons without a conscience, the performance is as displayed in figure 6.44.

The best classification is reached after 81 epochs, and is given in confusion matrix 6.15.

$$\begin{array}{cc} & \begin{array}{cc} A & E \end{array} \\ \begin{array}{c} A \\ \\ E \end{array} & \begin{pmatrix} 82.2 & 17.8 \\ & \\ 20.0 & 80.0 \end{pmatrix} \end{array} \tag{6.15}$$

6.4.7 Discrimination between A, B, C, D and E

This section deals with the problem of discriminating between all five classes simultaneously. Section 6.5 contains a discussion on these results.

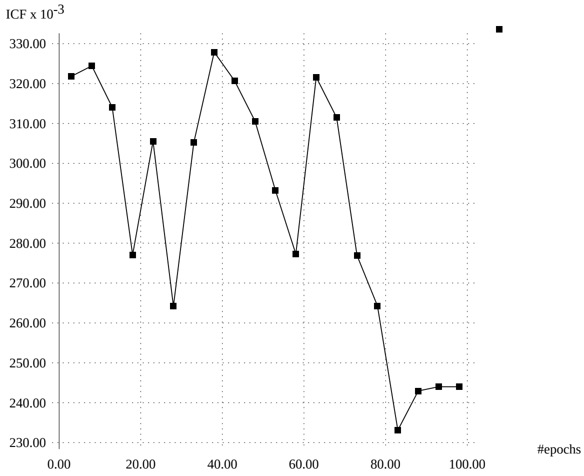


Figure 6.44: Performance on the AE-validation set for the optimal feature map.

Varying the architecture

The results of different architectures can be found in figures 6.45 to 6.46 for the training set, and in figures 6.47 to 6.48 for the test set.

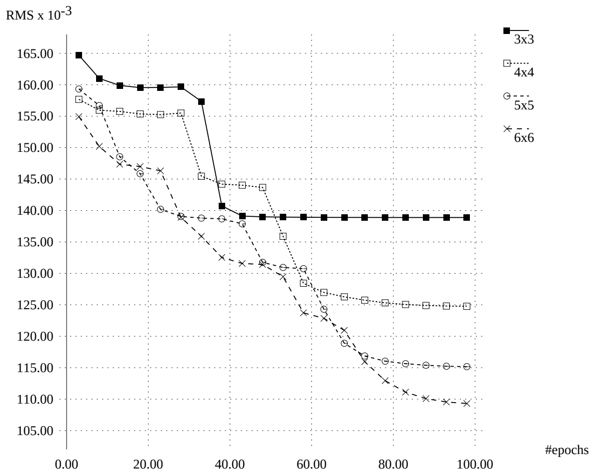


Figure 6.45: Performance on the ABCDE-training set for several architectures.

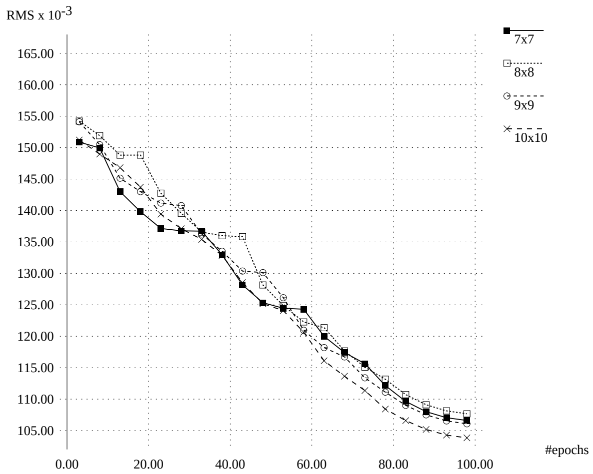


Figure 6.46: Performance on the ABCDE-training set for several architectures.

The best performance is achieved with a feature map with 7x7 neurons in the output layer.

Varying the learning rate

The influence on the performance of variations in the learning rate is presented in figure 6.49 for the training set, while figure 6.50 presents the performance on the test set.

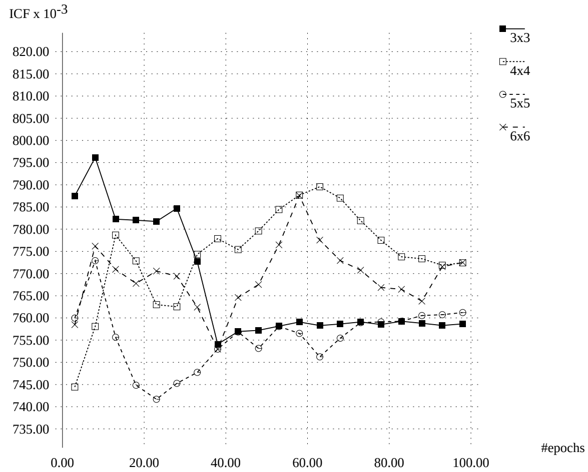


Figure 6.47: Performance on the ABCDE-test set for several architectures.

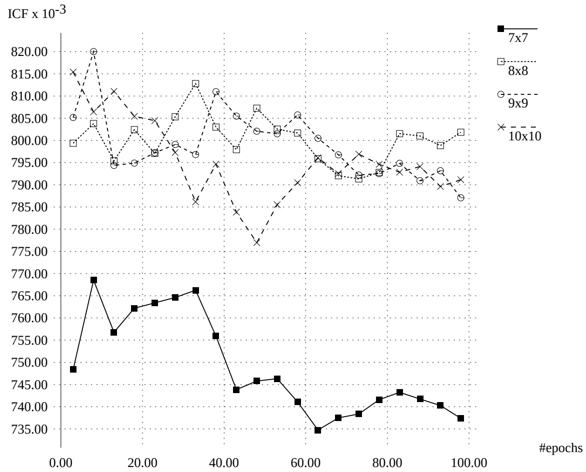


Figure 6.48: Performance on the ABCDE-test set for several architectures.

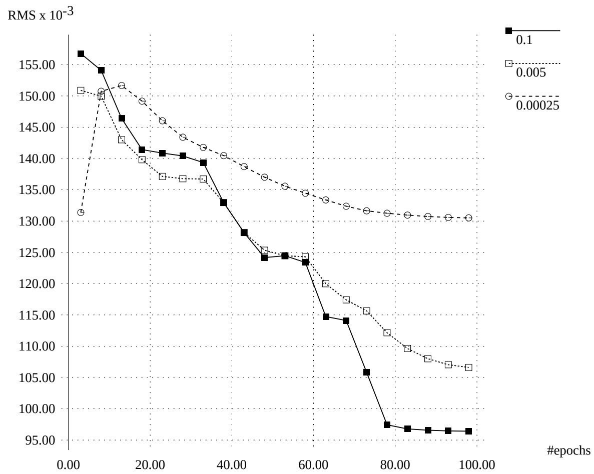


Figure 6.49: Performance on the ABCDE-training set for several learning rates.

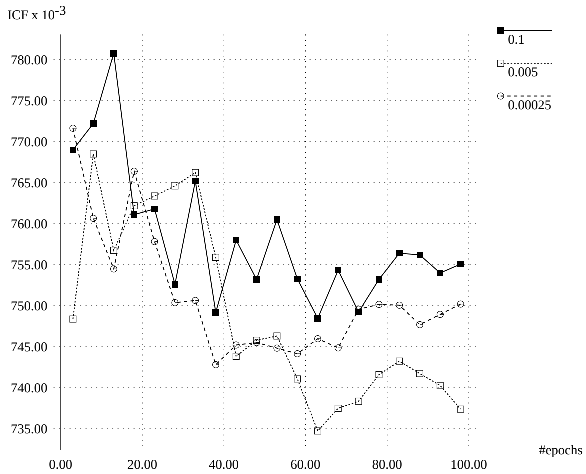


Figure 6.50: Performance on the ABCDE-test set for several learning rates.

The best choice for the learning rate is 0.005.

Adding conscience

If the neurons of the output layer are provided with a conscience, the performance on the training set is displayed by figure 6.51, while the performance on the test set is given by figure 6.52.

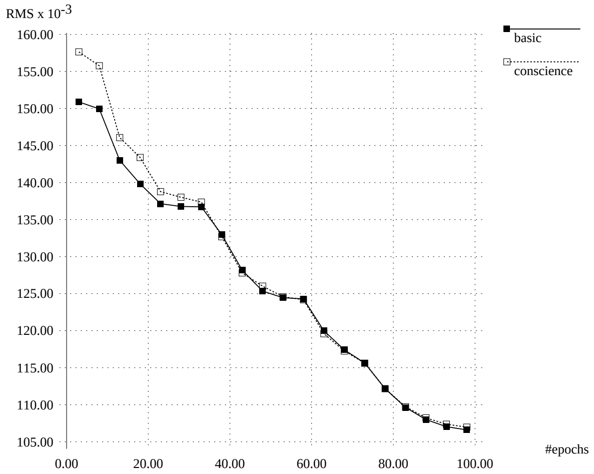


Figure 6.51: Performance on the ABCDE-training set using conscience.

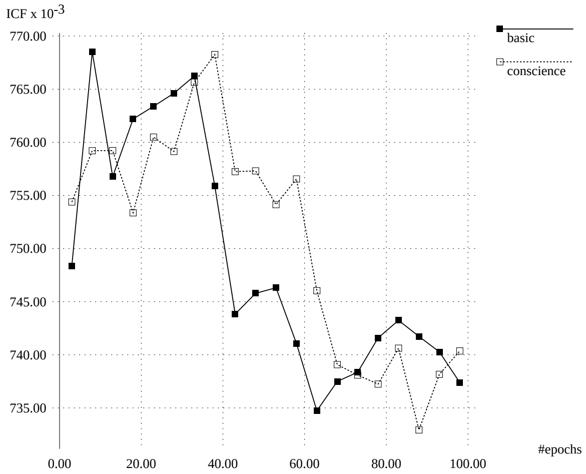


Figure 6.52: Performance on the ABCDE-test set using conscience.

The network with neurons with a conscience classifies the test set a fraction better than the basic network, and is therefore used for evaluation of the validation set.

Performance on the validation set

The performance of the 7x7 network with a learning rate of 0.005 and neurons with conscience can be found in figure 6.53.

The performance of this network is optimal after 26 epochs; the classification percentages are given in matrix 6.16.

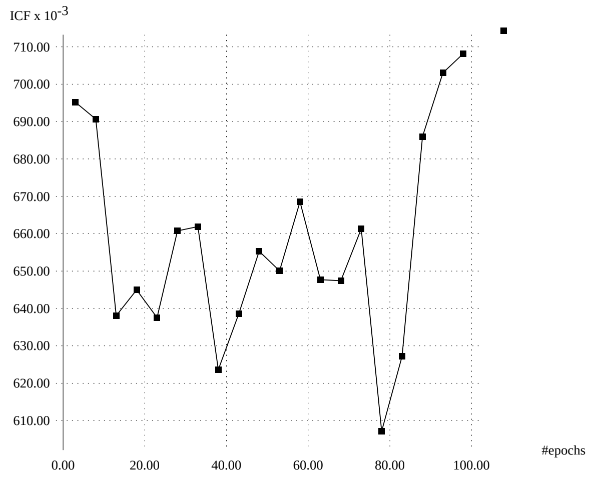


Figure 6.53: Performance on the ABCDE-validation set for the optimal feature map.

$$\begin{array}{cc} & \begin{array}{ccccc} \quad A & \quad B & \quad C & \quad D & \quad E \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} & \begin{pmatrix} 55.8 & 4.7 & 20.9 & 12.9 & 10.8 \\ 10.0 & 70.0 & 20.0 & 0.0 & 0.0 \\ 28.6 & 14.3 & 28.6 & 14.3 & 14.3 \\ 30.8 & 15.4 & 15.4 & 30.8 & 7.7 \\ 20.0 & 10.0 & 20.0 & 0.0 & 50.0 \end{pmatrix} \end{array} \tag{6.16}$$

6.5 Discussion

Looking at the figures of section 6.4 that display the *RMS* as a function of the number of epochs it strikes that the *RMS* does not decrease gradually, but in stages. This phenomenon, which is better visible at the smaller feature maps, is caused by the neighbourhood function. Although the neighbourhood radius decreases relatively gradual, the actual neighbourhood of a winning neuron decreases step by step. The reason for this behaviour is that the neurons of the output layer are organized in a grid — see figure 6.54 — and therefore have fixed distances to each other. As soon as the neighbourhood radius becomes less than a certain threshold, several neurons at the border of the neighbourhood radius are excluded from the set of neurons constituting the neighbourhood of a winning neuron. A good example of this behaviour can be found in figure 6.8, which displays performance of a 5x5 feature map with a learning rate of 0.005 and neurons without conscience on the AB-training set. In contrast with figure 6.10, which displays the same data but consists of only 20 points each being

2.8	2.2	2	2.2	2.8
2.2	1.4	1	1.4	2.2
2	1	0	1	2
2.2	1.4	1	1.4	2.2
2.8	2.2	2	2.2	2.8

Figure 6.54: Distances between neurons for a 5x5 feature map.

the average of five data points, this figure consists of 100 data points that are not averaged. As can be calculated[4], the stages in the *RMS* start at the epochs 12, 22, 45 and 62, excluding respectively the neurons at distance $\sqrt{5}$, 2, $\sqrt{2}$ and 1.

Another observation that can be made is that the *RMS* does not decrease much anymore at the end of the training process. A reason for this behaviour could be that the network has reached a minimum, but more likely is that the learning rate, which decreases according to equation 6.6, has reached a value that causes the weights to change only marginally. For the *ICF* the previous observation can not be made; evidently small changes in the network can cause big changes in the classification percentages. This suggests that the overlap of the different classes is considerable, something that has already been mentioned before when the jumpy behaviour of the *ICF* was explained in section 6.4.1.

In all figures where the size of the feature map is varied it can be found that the larger the feature map is the lower the *RMS* is. This is in agreement with the theory which states that a feature map can be trained to memorize a set of vectors to any accuracy by simply increasing the size of the map. Unfortunately the generalization abilities of the large networks are rather poor, something that can be seen in the figures displaying the *ICF* for the test set where the size of the feature map is varied. However, as can be found in the same figures, the size of the feature map should not be too small either. Although there does not seem to exist an order in which the feature maps between the smallest and the largest map — respectively 3x3 and 10x10 — have the best performance on the test set, the 7x7 map that lies between these two extremes often is the best choice.

[4]For instance, epoch 12 is calculated using equation 6.5 by $\frac{5/2-\sqrt{5}}{(5/2)/100} = 11$ where $5/2$ is the initial neighbourhood radius (see section 6.4.2) and 100 equals the number of iterations, so starting with epoch 12 neurons at a distance of $\sqrt{5}$ or more no longer belong to the neighbourhood of a winning neuron.

The figures that display the influence of the learning rate on the training process show that with a higher learning rate the network faster and better converges on the training set. The faster convergence is in accordance with the theory, the better convergence is not. A possible explanation for the bad convergence on the test set when using the smallest learning rate — 0.00025 — could be that the number of epochs is too small, for the network is updated in small steps, so many steps are needed to accomplish the same change in weights. However, when the network was trained ten times as long — 1000 epochs instead of 100 — convergence improved only a little. In theory the influence of the learning rate on the performance on the test set is not clear, but in practice choosing the best value is not too difficult. A value of 0.00025, given by the rule of thumb which states that a good value could be 0.25 divided by the number of patterns in the training set, gives the worst results. On the other hand a value of 0.1 usually seems to be too large, while for most problems a learning rate of 0.005 gives the best performance.

The shape of the *RMS* when using a learning rate of 0.00025 suggests that the initial distribution of the neurons over the feature space is rather good, because when the map is roughly organized and many neurons are translated over relatively large distances — this takes place in the start of the training process, see section 6.2.1 — the performance of the network on the training set gets worse, only improving again at the end of the training process when the network is tuned more finely to the training set. This phenomenon can not be observed when using a higher learning rate, because in this case the stage in which the map is roughly organized is much shorter because the neurons are translated over much larger distances.

The mechanism of providing each neuron with a conscience as described in section 6.3.1 does not improve the performance of the network on the training set. Evidently all neurons of the output layer represent some part of the training set, making the conscience of the neurons superfluous. On the test set this variation does not seem to give a structural improvement of the performance either.

The performance of the "optimal" network on the validation set is for all problems with only two classes over 80%, except for the problem of discriminating between classes A and C where the correct classification of class C is only 64.3%. These classes probably overlap more with each other than the other classes do. According to the confusion matrices 6.12 to 6.15 the order in which the classes can best be discriminated against class A is B, D/E and C. When discriminating between five classes simultaneously the order changes to B, A, E and C/D. The only difference between the two orders is that class D falls a place. While for all other classes the percentage of correct classification decreases 20 to 30%, the percentage of class D decreases about 45%. The additional 15% class D loses to class A: the percentage of patterns of class D that are classified as belonging to class A is doubled, while for the other classes this percentage stays the same or even decreases compared to the same percentage when discriminating between only two classes.

A last observation that is made is that the shapes of the *ICF* showing the performance of a feature map on the test set and the validation set are totally different. This could suggest that the test set is not representative for the validation set, but we assume that it is simply the result of the small number of patterns in the validation set which causes the *ICF* to react very strongly on the incorrect classification of just one more pattern. The fact that the achieved percentages of correct classification are rather high supports our confidence in the method by which we have generated the additional data.

For comparison with the other methods of classification — see section 7.2 — not only the classification percentages but also the computation time is of importance. For the worst problem that we have examined, a 10x10 feature map containing neurons without conscience to classify 2500 training patterns and 2500 test patterns, a Sun SPARCstation 1 needed 1 hour and 51 minutes.

Chapter 7

Conclusions

At the end of each chapter on discriminant analysis, back-propagation and feature mapping the results of the method under observation were discussed individually and conclusions were drawn. In this chapter however the results of all three methods are discussed at a higher level in order to answer the question posed in my graduation-assignment. The conclusions of this thesis can roughly be divided in two parts: the first set of conclusions — see section 7.1 — deals with the use of ultrasonic tissue characterization on our specific application, using the data set that is at our disposal, while the other subject — see section 7.2 — is more of computer scientific nature, being the comparison of the methods that have been investigated. Finally some recommendations for future investigations are made in section 7.3.

7.1 Applicability of UTC on our data

Nowadays at the Biophysics Laboratory discriminant analysis is used to classify patients based on the principle of ultrasonic tissue characterization. For the type of diffuse liver diseases that we have examined, using the five parameters described in section 3.2, discriminant analysis applying cross-validation resulted in an unsatisfactory performance for most of the small classes, see for instance matrix 4.36 or 4.43. If instead of cross-validation resubstitution is used, the percentage of correct classification for these classes enormously increases, without decreasing the percentage of correct classification for the large class. Obviously a good partitioning is possible, only it can not be found because the original data set is much too small to extract the general characteristics of the classes from it.

But there are more reasons to extend the original data set with generated cases. One ground is that in case of such small classes as for our problem the determination of the performance is very rough: if the classification of only one patient belonging to one of the smallest classes changes, the percentage of correct classification for that class changes with 10%. Even if the original data set would contain some general characteristics for each class, it would be impossible to find the optimal partitioning of the feature space, because a small change in the positioning of the borders between the classes does not necessarily imply a change in performance, although it could be a significant change. Another motivation for the extension of the data set is that we want to use neural networks for the classification of patients. Usually when dealing with neural networks the data set is classwise split into a training and a validation set using a 2:1 ratio; for a class containing only 10 patients this would mean a training set of 7 and a validation set of 3 patients! The problems described previously in this section apply to this situation even more.

The method we have used for the generation of additional data — the kernel based method, see section 3.4.2 — results in data that can well be used for our purposes. A large training set has been generated on the basis of which the feature space is partitioned, after which an equally large test set has been used to check whether the partitioning follows the general characteristics of the classes or is based on features that are coincidentally present. Especially for back-propagation the test set has proven to be very useful for the detection of over-learning. The performance on the validation set that is achieved using the partitioning acquired in this manner is in every respect acceptable compared to the results using discriminant analysis with cross-validation. It could be argued that a good performance on the validation set is obvious, since the training set has been generated on the basis of the validation set and thus is fully correlated with it, but this is not true. For the performance on the validation set of discriminant analysis using the training set to partition the feature space is for all problems worse than the performance on the original data of discriminant analysis using resubstitution, which corresponds to discriminant analysis using a training and a validation set that are completely correlated.

For the performance of the individual classes holds that good results can be achieved, if there is to be discriminated between only two classes at the same time. In this case the percentage of correctly classified patients is for all classes over 80%, while for some classes even 90% or 100% can be reached. If there is to be discriminated between all five classes simultaneously, the performance deteriorates for all methods, but for discriminant analysis and feature mapping the drop in performance is much higher than for back-propagation. While for discriminant analysis the minimum percentage of correct classification is 7.7%, for back-propagation this is 46.2%. Although little can be said about the order in which the classes can best be discriminated, it strikes that for all methods class A — the healthy people — and class B — people suffering from primary biliary cirrhosis — score percentages of correct classification that are significantly higher than those of the other classes, while class D — people suffering from acute hepatitis — usually has the lowest percentage. Evidently classes A and B are situated relatively isolated in the feature space or contain patterns that are rather similar, making a good discrimination possible, while the other classes and especially class D are much more scattered.

Considering all the facts summarized in this section we come to the conclusion that classification of patients on the basis of the parameters and for the diseases that we have selected is impossible because the original data set is much too small. However, if the data set is extended with some generated cases based on the original data, good results can be achieved.

method	problem	class A	class B	class C	class D	class E
Discriminant analysis	AB	89.8%	100.0%	–	–	–
Feature mapping	AB	94.6%	90.0%	–	–	–
Back-propagation	AB	93.8%	100.0%	–	–	–
Discriminant analysis	AC	80.6%	–	64.3%	–	–
Feature mapping	AC	80.6%	–	64.3%	–	–
Back-propagation	AC	81.4%	–	92.9%	–	–
Discriminant analysis	AD	83.0%	–	–	69.2%	–
Feature mapping	AD	81.4%	–	–	84.6%	–
Back-propagation	AD	86.0%	–	–	84.6%	–
Discriminant analysis	AE	77.5%	–	–	–	80.0%
Feature mapping	AE	82.2%	–	–	–	80.0%
Back-propagation	AE	92.2%	–	–	–	80.0%
Discriminant analysis	ABCDE	64.4%	70.0%	35.7%	7.7%	30.0%
Feature mapping	ABCDE	55.8%	70.0%	28.6%	30.8%	50.0%
Back-propagation	ABCDE	80.6%	80.0%	64.3%	46.2%	60.0%

Table 7.1: Percentage correctly classified patients for the different methods.

7.2 Comparison of the investigated methods

In the chapters 4 to 6, dealing with the different methods, we have already made clear that a fair comparison between the methods can be performed, since all methods use exactly the same data sets and the final results are always presented using confusion matrices. Also the line of approach is for all three methods the same: first the feature space is partitioned using the training set, then, except for discriminant analysis, the ability to generalize is checked using the test set, and finally the performance on the validation set is assessed using the acquired partitioning.

For the comparison of the methods we take several aspects of the methods into consideration, of which the most important one of course is the performance on the validation set. In table 7.1 the percentage of correct classification for each class of the validation set for our five problems have been gathered for the investigated methods. For discriminant analysis the results have been obtained using the quadratic variant, while the feature space has been partitioned using the generated data; the results of back-propagation and feature mapping have been achieved by the optimal network for each problem, see sections 5.4.3 to 5.4.7 and 6.4.3 to 6.4.7. From table 7.1 it can be concluded that the method that gives the best results is back-propagation, since it achieves the highest percentages of correct classification for all problems; especially when dealing with five classes at the same time the difference in performance is very clear. Between the other two methods — discriminant analysis and feature mapping — there is not much difference, although for some problems feature mapping is a fraction better.

Another aspect of the methods that we look at is the computation time, divided in the time needed to partition the feature space — train the network, calibrate the feature space — and the time needed to classify one patient. Although it usually is difficult to measure the time needed to solve a problem because time depends on the power of the computer used, the current load of the computer etcetera, in our case it is simple. For the time needed to partition the feature space is of different orders for the different methods: discriminant analysis needs approximately half a minute, feature mapping needs two hours, and back-propagation needs six days. The long time to train the networks is of course a nuisance, but it is something that only has to be done once. If the network reaches a state in which its performance is good enough, the entire network can be saved on some kind of background memory or even be put on some sort of dedicated chip. Furthermore, at the moment there is a lot of research into the ability of neural networks to learn *incrementally*; this entails that if the training set is extended with new patients, the neural network does not have to be trained all over again, but a limited number of epochs with the new training set will suffice. If neural networks can learn incrementally, this would reduce the computation time in the future. The time needed to classify one patient, which is far more important for the application of ultrasonic tissue characterization in a practical situation, is for each of the investigated methods very short, being less than a second.

A final aspect that we want to consider is the temporal behaviour of the method, something which only applies to neural networks. Both feature mapping and back-propagation display an equally gradual convergence of the network on the training set, but on the test as well as the validation set back-propagation shows a relatively smooth performance curve while the performance curve of a feature mapping network is much more irregular. Although a gradual change in the performance is preferred over an irregular one, this does not mean that a feature mapping network is useless for this type of application; it only indicates that the number of epochs that the feature map is to be trained should be carefully chosen.

Taken all the facts presented in this section into account, we conclude that of the three methods that we have investigated back-propagation is with length the best choice for our application.

7.3 Suggestions for future research

If ultrasonic tissue characterization is ever to be used as a diagnostic tool for the clinician, the number of diseases that can simultaneously be discriminated must be extended. This would probably mean that a lot more supervised data must be available for training purposes, something that has already proven to be a problem. We therefore recommend that the acquisition of data continues, maybe even in collaboration with other hospitals[1].

If the data set is large enough, the biggest factor of unreliability in this thesis, the generation of data, can be eliminated. Until then, the generation of data is a necessary evil whose influence on the results must be kept as limited

[1] In this case the equipment needed for the acquisition has to be adjusted to some standard.

as possible. It may be a good idea to investigate more thoroughly the statistical correctness of the data generation method used, or of the many other methods available for the generation of data, since the data set is the basis on which the entire research of this application is built.

Another interesting point that could be investigated is the influence of selecting different parameters or a different number of parameters on the classification results. The choice of the parameters that have been selected in this thesis is based on similar research performed for discriminant analysis, but it could well be possible that for neural networks other or more parameters result in a better performance.

In this thesis only three methods have been investigated, but of course there exist many more, both in the field of statistics as well as neural networks. As a possible statistical method we think of Principal Component Analysis, while as neural network one of the models based on the Adaptive Resonance Theory seems to be very suitable. On problems similar to our problem ARTMAP has shown to yield good results.

Bibliography

[1] Oosterveld, B.J., *On the Quantitative Analysis of Ultrasound Signals with Applications to Diffuse Liver Disease*, University of Nijmegen, Nijmegen, Ph.D. project report, 1990.

[2] Nikoonahad, M. and Liu, D.C., *Medical Ultrasound Imaging using Neural Networks*, Electronics Letters 26, 545-546, 1990.

[3] Parikh, J.A. and DaPonte, J.S., *Application of Neural Networks to Pattern Recognition Problems in Remote Sensing and Medical Imagery*, Applications of Artificial Neural Networks SPIE 1294, 146-160, 1990.

[4] Silverman, R.H. and Noetzel, A.S., *Image Processing and Pattern Recognition in Ultrasonograms by Backpropagation*, Neural Networks 3, 593-603, 1990.

[5] Ostrem, J.S., Valdes, A.D. and Edmonds, P.D., *Application of Neural Nets to Ultrasound Tissue Characterization*, Ultrasonic Imaging 13, 298-299, 1991.

[6] Refenes, A.N. and Bilge, U., *Self-Organizing Feature Maps in Pre-Processing Datasets for Decision Support in Histopathology*, NSC/BME-90E, Proc. Int. North Sea Conf. in Biomedical Engineering, 1990.

[7] Refenes, A.N., Jain, N. and Alsulaiman, M.M., *An Integrated Neural Network System for Histological Image Understanding*, Proc. Int. Symp. SPIE, 1990.

[8] Leidse Onderwijs Instellingen, *Basiskennis Echografie. Vak 902*, Nederlandse Vereniging voor Ultrageluid in de Geneeskunde en de Biologie, 1992.

[9] Verhoeven, J.T.M., *Een Snelle Digitale Signaal Verwerker voor het Ultrasoon Biopsie Apparaat*, Technical University of Eindhoven, Eindhoven, graduation project report, 1986.

[10] Fabel, R., *Laterale Diffraktie-Korrektie van Ultrageluidsscans*, Technical University of Eindhoven, Eindhoven, graduation project report, 1988.

[11] Goodman, J.W., *Some fundamental properties of speckle*, J Optical Society of America 66, 1145-1150, 1976.

[12] Goodman, J.W., *Statistical Optics*, John Wiley & Sons, New York, 1975.

[13] Wagner, R.F., Smith, S.W., Sandrik, J.M. and Lopez, H., *Statistics of Speckle in Ultrasound B-Scans*, IEEE Transactions on Sonics and Ultrasonics 30, 156-163, 1983.

[14] Thijssen, J.M. and Oosterveld, B.J., *Texture in Tissue Echograms. Speckle or Information?*, J Ultrasound Medicine 9, 215-229, 1990.

[15] Oosterveld, B.J., Thijssen, J.M. and Verhoef, W.A., *Texture of B-Mode Echograms: 3-D Simulations and Experiments of the Effects of Diffraction and Scatterer Density*, Ultrasonic Imaging 7, 142-160, 1985.

[16] Holmström, L. and Koistinen, P., *Using Additive Noise in BackPropagation Training*, IEEE Transactions on Neural Networks 3, 24-38, 1992.

[17] Duin, R.P.W., *On the Choice of Smoothing Parameters for Parzen Estimators of Probability Density Functions*, IEEE Transactions on Computers C-25, 1175-1179, 1976.

[18] Schikhof, W., *Analyse en Calculus 2*, University of Nijmegen, Nijmegen, reader, 1984.

[19] Albert, A and Harris, E.K., *Multivariate Interpretation of Clinical Laboratory Data*, Marcel Dekker, Inc., New York, 1987.

[20] SAS Institute Inc., *SAS/STAT User's Guide Version 6 Volume 1*, SAS Institute Inc., Cary (NC), 1989.

[21] Dayhoff, J.E., *Neural Network Architectures. An Introduction*, Van Nostrand Reinhold, New York, 1990.

[22] Hertz, J., Krogh, A. and Palmer, R.G., *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company, Redwood City, 1991.

[23] Nowlan, S.J. and Hinton, G.E., *Soft Weight-Sharing*, University of Toronto, Toronto, technical report, 1991.

[24] Bierman, S. and Hooft, H. van, *A Kohonen Neural Networks Simulator*, University of Nijmegen, M.Sc. project report in preparation.

Appendix A

Software

This chapter contains short user manuals for the various programs written for this thesis. At the end of the chapter the manual page about the changes in the Kohonen Neural Networks Simulator — called koho — by S. Bierman and H. van Hooft can be found.

The programs for which the user manual or, in the case of koho, the additional manual page is included are:

- classify
- shuffle
- kernelgen
- statgen
- merge
- format
- histogram
- analyze
- plot
- bp
- koho

For a more detailed description of these programs is referred to the commentary in the source code, while more information about the Kohonen Neural Networks Simulator can be found in [24].

NAME

classify - select patients and parameters from a database

SYNOPSIS

classify class

DESCRIPTION

classify selects all patients belonging to class and saves them in a file. For every patient the parameters MU, B, SNR, S and A1 are saved in that order, preceded by the patient number and a new class identification. The first line of the new file contains the number of patients followed by the number of parameters, which in this case is 5.

OPTIONS

For class the following options are available:

A Select patients with database classification 1. Their new classification is 0.

B Select patients with database classification 28, 281, or 282. Their new classification is 1.

C Select patients with database classification 22, 23, 24, 26 or 27. Their new classification is 2.

D Select patients with database classification 20 or 21. Their new classification is 3.

E Select patients with database classification 25. Their new classification is 4.

FILES

~/data/original/parameters0.txt
first part of patient database

~/data/original/parameters1.txt
second part of patient database

~/data/A/org new file for class A

~/data/B/org new file for class B

~/data/C/org new file for class C

~/data/D/org new file for class D

~/data/E/org new file for class E

SEE ALSO

shuffle(n), kernelgen(n), statgen(n), merge(n), format(n), histogram(n)

BUGS

The program is not very flexible, since the file names, the new class indices and the selected parameters can not be changed without recompilation. However, all these dependencies can easily be changed because the program is written using #define-statements.

If the number of parameters is different from 5 or a new class is to be added, the source code has to be changed in a few places after which the program has to be recompiled.

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

SHUFFLE(n) MISC. REFERENCE MANUAL PAGES SHUFFLE(n)

NAME
shuffle - shuffle the order of patients in a file

SYNOPSIS
shuffle <u>file</u>

DESCRIPTION
shuffle reads the patients contained in <u>file</u> and writes them in a random order to the same file. The first line of <u>file</u> should contain the number of patients followed by the number of parameters, which corresponds the output generated by the program classify. This information is incorporated in the shuffled version of the file as well.

SEE ALSO
classify(n), kernelgen(n), statgen(n), merge(n), format(n), histogram(n)

AUTHOR
Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

kernelgen - generate new patients using the kernel based method

SYNOPSIS

kernelgen class

DESCRIPTION

kernelgen generates new patients for class adding up to 1000 patients. The algorithm that is the basis of this program has been designed by Holstrom and Koistinen, and can be found in IEEE trans. Neural Networks 3, p. 24-38, 1992. The Bartlett random generator is implemented using the method of Newton with a precision of 0.005. The new patients are also provided with an unique patient number, depending on their class. For every parameter of the patients the mean and standard deviation are displayed. The first line of the input file of kernelgen, see FILES, should contain the number of patients followed by the number of parameters. This corresponds to the output generated by the programs classify and shuffle. The first line of the output file, see FILES, also contains the number of patients followed by the number of parameters.

OPTIONS

For class the following options are available:

A Generate new patients for class A. The new patient numbers start with 5000.

B Generate new patients for class B. The new patient numbers start with 1000.

C Generate new patients for class C. The new patient numbers start with 2000.

D Generate new patients for class D. The new patient numbers start with 3000.

E Generate new patients for class E. The new patient numbers start with 4000.

FILES

~/data/A/use	input file for class A
~/data/A/gen	output file for class A
~/data/B/use	input file for class B
~/data/B/gen	output file for class B
~/data/C/use	input file for class C

~/data/C/gen	output file for class C
~/data/D/use	input file for class D
~/data/D/gen	output file for class D
~/data/E/use	input file for class E
~/data/E/gen	output file for class E

SEE ALSO

classify(n), shuffle(n), statgen(n), merge(n), format(n), histogram(n)
L. Holmstrom and P. Koistinen, Using Additive Noise in Back-Propagation Training

BUGS

The program is not very flexible, since the file names, the start of the new patient numbers, the total number of patients and the precision can not be changed without recompilation. However, all these dependencies can easily be changed because the program is written using #define-statements.

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

statgen - generate new patients using a statistical method

SYNOPSIS

statgen _class_

DESCRIPTION

statgen generates new patients for _class_ to a total number of patients of 1000. The algorithm that is the basis of this program has been described by Maurice klein Gebbinck in Ultrasonic Tissue Characterization Using Neural Networks, the University of Nijmegen, M.Sc. project report, 1992. The segment out of which new patients are drawn has a length of 4 SD, 2 SD to the left and 2 SD to the right of the mean for each parameter. The new patients are also provided with an unique patient number, depending on their class. For every parameter of the patients the mean and standard deviation are displayed. The first line of the input file of kernelgen, see FILES, should contain the number of patients followed by the number of parameters. This corresponds to the output generated by the programs classify and shuffle. The first line of the output file, see FILES, also contains the number of patients followed by the number of parameters. For _class_ the following options are available:

A Generate new patients for class A. The new patient numbers start with 5000.

B Generate new patients for class B. The new patient numbers start with 1000.

C Generate new patients for class C. The new patient numbers start with 2000.

D Generate new patients for class D. The new patient numbers start with 3000.

E Generate new patients for class E. The new patient numbers start with 4000.

FILES

~/data/A/use	input file for class A
~/data/A/owngen	output file for class A
~/data/B/use	input file for class B
~/data/B/owngen	output file for class B
~/data/C/use	input file for class C
~/data/C/owngen	output file for class C

~/data/D/use	input file for class D
~/data/D/owngen	output file for class D
~/data/E/use	input file for class E
~/data/E/owngen	output file for class E

SEE ALSO

classify(n), shuffle(n), kernelgen(n), merge(n), format(n), histogram(n)
M. klein Gebbinck, Ultrasonic Tissue Characterization Using Neural Networks

BUGS

The program is not very flexible, since the file names, the start of the new patient numbers, the total number of patients, the correlation coefficients and the length of the segment out of which the patients are drawn can not be changed without recompilation. However, all these dependencies can easily be changed because the program is written using #define-statements.

If the number of parameters is different from 5, the source code has to be changed in a few places after which the program has to be recompiled to incorporate a different number of correlation coefficients.

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

merge - merge several patient files into one

SYNOPSIS

merge <u>problem</u>

DESCRIPTION

merge reads the patients of several files and places them in a new file. The input files, see FILES, should all have a first line containing the number of patients and the number of parameters. This file format conforms to the output of the programs kernelgen and statgen. merge checks whether all input files contain the same number of parameters and aborts if this is not the case. After merging, the first line of the output file, see FILES, contains the number of patients and the number of parameters.

OPTIONS

For <u>problem</u> the following options are available:

AB	Merge classes A and B.
AC	Merge classes A and C.
AD	Merge classes A and D.
AE	Merge classes A and E.
ABCDE	Merge classes A, B, C, D and E.

FILES

~/data/A/gen	input file for problems AB, AC, AD, AE and ABCDE
~/data/B/gen	input file for problems AB and ABCDE
~/data/C/gen	input file for problems AC and ABCDE
~/data/D/gen	input file for problems AD and ABCDE
~/data/E/gen	input file for problems AE and ABCDE
~/data/problems/AB/gen	output file for problem AB
~/data/problems/AC/gen	output file for problem AC
~/data/problems/AD/gen	output file for problem AD
~/data/problems/AE/gen	output file for problem AE
~/data/problems/ABCDE/gen	output file for problem ABCDE

SEE ALSO

classify(n), shuffle(n), kernelgen(n), statgen(n), format(n), histogram(n)

BUGS

The program is not very flexible, since the file names can not be changed without recompilation. However, they can easily be changed because the program is written using #define-statements.

If a new problem is to be added, the source code has to be changed in a few places after which the program has to be recompiled.

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

format - scale patient parameters and add target patterns

SYNOPSIS

format _problem_ _target_

DESCRIPTION

format scales all parameters of the patients of _problem_ between 0.1 and 0.9. For each parameter the mean minus 2 SD is projected to 0.1 and the mean plus 2 SD to 0.9, while intermediate values are scaled linearly in this interval; values lower (higher) than the mean minus (plus) 2 SD are projected to 0.1 (0.9). Furthermore, each patient is provided with a target pattern, depending on the _target_. The target pattern is nothing more than a representation of the class a patient belongs to that is needed when dealing with neural networks. The first line of the input file, see FILES, should contain the number of patients and the number of parameters. This file format is conform the output of the program merge. The information contained in the first line of the output file, see FILES, is also dependent on the _target_.

OPTIONS

For _problem_ the following options are available:

AB Format problem AB.

AC Format problem AC.

AD Format problem AD.

AE Format problem AE.

ABCDE Format problem ABCDE.

For _target_ the following options are available:

BACKPROP The first line of the output file contains the number of patients, the number of parameters and the number of classes. The target pattern of a patient is placed immediately below the data of the corresponding patient and consists of a vector containing one element with a high value, while all other elements have low values. The patient number is saved as well.

SIMPLEKOHO The first line of the output file contains a zero, followed by the number of parameters and the number of patients. Since this option corresponds to unsupervised Kohonen, no target pattern is supplied.

SUPERKOHO
: The first line of the output file contains an one, followed by the number of parameters and the number of patients. The target pattern of a patient is placed immediately before the data of the corresponding patient and consists of a single class number.

FILES

~/data/problems/AB/gen	input file for problem AB
~/data/problems/AB/pat	output file for problem AB
~/data/problems/AC/gen	input file for problem AC
~/data/problems/AC/pat	output file for problem AC
~/data/problems/AD/gen	input file for problem AD
~/data/problems/AD/pat	output file for problem AD
~/data/problems/AE/gen	input file for problem AE
~/data/problems/AE/pat	output file for problem AE
~/data/problems/ABCDE/gen	input file for problem ABCDE
~/data/problems/ABCDE/pat	output file for problem ABCDE

SEE ALSO

classify(n), shuffle(n), kernelgen(n), statgen(n), merge(n), histogram(n)

BUGS

The program is not very flexible, since the file names and the range of the new scaling interval can not be changed without recompilation. However, all these dependencies can easily be changed because the program is written using #define-statements.

If a new problem is to be added, the source code has to be changed in a few places after which the program has to be recompiled.

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

histogram - create a histogram of a patient file suitable for xgraph

SYNOPSIS

histogram

DESCRIPTION

histogram is an interactive program that, for every parameter a patient file consists of, creates a histogram file suitable for xgraph. After starting the program, first it is prompted for the name of the patient file containing the data for the histograms. The first line of the patient file should contain the number of patients followed by the number of parameters. This file format corresponds to the output of the programs classify, kernelgen and statgen. Next, it is asked how many bars each histogram should consist of; a minimum of 3 bars is required. The next choice that has to be made is whether the histograms should range from the minimum value to the maximum value, or from the mean minus 2 SD to the mean plus 2 SD. If the SD-range is chosen, the leftmost (rightmost) bar of each histogram represents the number of patients with a value less (greater) than the mean minus (plus) 2 SD. Depending on the range, the minimum and maximum or the mean value and standard deviation appear at the standard output for each parameter. The file names the histograms are written to consist of two parts: a prefix and a parameter-dependent suffix; the prefix will be prompted for only once, the suffix once for each parameter.

SEE ALSO

classify(n), shuffle(n), kernelgen(n), statgen(n), merge(n), format(n), histogram(n), xgraph(l)

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

analyze - analyze a neural network performance file to determine the optimal epoch

SYNOPSIS

analyze file number-of-classes

DESCRIPTION

analyze compares the results of the neural network of each epoch listed in file and searches the optimal epoch. Every line in file should contain an epoch number, followed by the root mean squares (RMS) and the incorrect classification fractions (ICF), and, for every class, the number of erroneously classified patients, all of the training set. The number-of-classes should be equal to the actual number of classes in file. The rest of the line should contain the RMS, ICF and, again for every class, the number of erroneously classified patients of the test set. This file format corresponds to the output of the neural network simulators bp and koho. An epoch is found to be optimal if it has the lowest ICF for the test set in the file. If there exist other epochs with an equally low ICF, it has among these the lowest RMS for the training set. If there still remain several epochs, the epoch with the lowest number is chosen. The output is written to the standard output.

SEE ALSO

plot(n), bp(n), koho(n)

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

plot - create a plot of a neural network performance file suitable for xgraph

SYNOPSIS

plot

DESCRIPTION

plot is an interactive program that creates a polygon file suitable for xgraph. After starting the program, first it is prompted for the name of the file where plot should write the polygon plot to. Hereafter the choice must be made whether a plot of the results of the training or the test set is to be made, using the root mean squares (RMS) or the incorrect classification fractions (ICF). Next, the number of classes contained in the performance file that is to be plotted must be given, followed by the highest epoch number in the performance file. The next question concerns the number of points the polygon sould consist of, and the number of epochs that should be averaged to constitute one point of the polygon. It is obvious that the number of available epochs puts some restrictions on the number of points and the number of epochs to be averaged. After all previous input has been validated, the user will repeatedly be prompted for the name of a performance file to be plotted and its corresponding name for the legend. After every plot it is asked if another plot is to be added. Every line in the performance file should contain an epoch number, followed by the total sum of squares (RMS) and the incorrect classification fractions (ICF), and, for every class, the number of erroneously classified patients, all of the training set. The earlier input number-of-classes should be equal to the actual number of classes in the performance file. The rest of the line should contain the RMS, ICF and, again for every class, the number of erroneously classified patients of the test set. This file format corresponds to the output of the neural network simulators bp and koho.

SEE ALSO

analyze(n), bp(n), koho(n)

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME
bp - a back-propagation neural networks simulator

SYNOPSIS
bp

DESCRIPTION
bp is an interactive program that simulates a neural network based on the back-propagation algorithm. After starting the program, the main menu is displayed. The following menu items are available:

create network...
Create a new network from scratch. If another network has already been defined, it is asked whether or not to destroy the current network. The parameters that have to be filled in are the lower and upper boundary of the weights, the number of layers including the input and output layer, and for each layer the number of neurons.

load network...
If a network has been saved with bp previously, it can be loaded using this command. If another network has already been defined, it is asked whether or not to destroy the current network. For the filename of the network is prompted.

save network...
Save the current network. For the filename of the network is prompted; if the file already exists, it is asked whether or not to overwrite that file.

load configuration...
If a configuration has been saved with bp previously, it can be loaded using this command. Since the program starts with a default configuration, it is always asked whether or not to overwrite the current configuration. The configuration consists all parameters except the architecture and weights of the network, and the currently loaded patterns. For the filename of the network is prompted.

save configuration...
Save the current configuration. For the filename of the configuration is prompted; if the file already exists, it is asked whether or not to overwrite that file.

load learn patterns...
Load the patterns with which the network is trained; if another set of learn patterns has already been loaded, it is asked whether or not to overwrite these patterns. The format of the pattern file should be conform to the

output of the program format where the target is chosen to be BACKPROP. This entails a first line containing the number of patterns in the file, the dimension of the input vector and the dimension of the target vector. Each patient takes up two lines: the first line contains the patient number followed by the input vector, while the second line contains the target vector. For the filename of the patterns is prompted.

load test patterns...
For this command accounts the same as for load learn patterns..., except that the network uses these patterns only for recalling. This mechanism can be used to detect over-learning.

learn...
This is the main command of the program; here the configuration is changed and learning is initiated. Before the network can start learning, a network must have been created or loaded, and training patterns must be available in the right format. This implies that the dimension of the input (target) vector must be equal to the number of neurons in the input (output) layer. Loading of test patterns is optional. Many parameters have to be filled in. First of all the error function has to be chosen; the common squares can be selected, but also weight decay can be chosen. If the error function is set at weight decay, a value for gamma has to be input. The next choice is that of the activation function, sigmoidal or hyperbolic tangent can be chosen. Now the learn mode must be selected: by pattern, update the weights after each pattern, or by epoch, where the weights are updated after all patterns have been presented. If learning by pattern is chosen, the user will be prompted for the learn mode. The order in which the patterns are presented can either be permuted or round robin (cyclic). Other parameters for which values are to be given are the learning rate, the momentum and the maximum noise percentage. The maximum noise percentage determines the maximum amount of noise that is added to the training patterns; the percentage doesn´t imply an amount relative to the value of the input vector, but an absolute amount expressed in units of one hundredth. Next the number of epochs that the network is to be trained is asked, followed by the show and save frequencies. The show frequency determines the number of epochs after which the performance of the network is saved to the performance file, while the save frequency determines the number of epochs after which both the network as well as the configuration are saved to respectively the files network.save and configuration.save. Finally for the filename of the

performance file is prompted; if the file already exists, it is asked whether or not to overwrite that file. The format of the performance file starts with the epoch number, followed by the root mean squares (RMS) and the incorrect classification fractions (ICF), and, for every class, the number of erroneously classified patients, all of the training set. The rest of the line should contain the RMS, ICF and, again for every class, the number of erroneously classified patients of the test set. The resulting performance file can be analyzed and plotted using the programs analyze and plot.

show misclassified learn patterns
Show all learn patterns where the classification based on the output vector doesn´t agree with the classification based on the target vector. The output is written to the standard output. Of course a network and learn patterns in the right format have to be available.

show misclassified test patterns
For this command accounts the same as for show misclassified learn patterns.

quit Quit bp; if a network has been defined, it is asked whether or not it can be deleted.

SEE ALSO

analyze(n), plot(n)
M. klein Gebbinck, <u>Ultrasonic Tissue Characterization Using Neural Networks</u>

BUGS

If a different network (patterns) is to be loaded while a network (patterns) already exists, and the filename of the network (patterns) to be loaded is invalid, no network (patterns) are available anymore.

It shouldn´t be possible to load patterns before a network is available, because the format of the patterns has to agree with the architecture of the network. Now for several commands the same checks have to be performed.

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

NAME

koho - a Kohonen neural networks simulator

SYNOPSIS

koho

DESCRIPTION

koho is in fact nothing more than a shell script that starts up the program iface and sequent, and creates pipes that these programs use to communicate. iface is an ascii-based interface for the user to be able to communicate with the actual simulator sequent. In these manual pages only the changes in iface and sequent are discussed, for more information on the neural networks simulator is referred to the master thesis in preparation called A Kohonen Neural Networks Simulator by Silvio Bierman and Hans van Hooft.

Apart from the many bugs that have been resolved, the program has been altered to incorporate a neighbourhood that linearly decreases to zero with the number of epochs. A second alteration concerns the monitoring of the performance of the neural network. Now, after selecting the learning menu, the user is prompted for the name of the file the simulator writes the performance results to. The format of the performance file is equal to that of the back-propagation simulator bp. This implies a file where every line starts with an epoch number, followed by the root mean squares (RMS) and the incorrect classification fractions (ICF), and, for every class, the number of erroneously classified patients, all of the training set. The rest of the line should contain the RMS, ICF and, again for every class, the number of erroneously classified patients of the test set. This last change has a negative influence on the speed of the simulator. The resulting performance file can be analyzed and plotted using the programs analyze and plot.

SEE ALSO

analyze(n), plot(n)

S. Bierman and H. van Hooft, A Kohonen Neural Networks Simulator

AUTHOR

Maurice klein Gebbinck <mauricek@cs.kun.nl>

Appendix B

Erratum

After this report had been sent to the copy-shop, it was discovered that the program which had been used to generate data — kernelgen — contained a minor but significant error. Instead of randomly selecting an original patient for each patient to be generated, an original patient was selected for each parameter of each patient to be generated. This strategy led to complete loss of correlation between the parameters, as can be found in section 3.4.2.

Tables B.1 to B.5 display the statistics of the generated data using the corrected data generation program. Comparison of these tables to tables 3.1 to 3.5 show that for all parameters the mean, standard deviation and correlation with the other parameters is maintained within certain limits. For this new data sets the standard deviation is structurally higher and the correlation is structurally lower, which suggests a smaller value for the smoothing parameter should be chosen. However, this would cause the artificial data to be more dependent on the original data.

After formatting this new data according to the method described in section 3.5, the results of quadratic discriminant analysis using a calibration set, when there is to be discriminated between five classes simultaneously, can be found in matrices B.1 to B.3 for respectively the calibration, test and validation set.

$$
\begin{array}{c c}
 & \begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array} &
\left(\begin{array}{rrrrr}
82.4 & 4.2 & 3.6 & 1.2 & 8.6 \\
2.4 & 76.8 & 3.6 & 7.0 & 10.2 \\
18.8 & 12.6 & 42.4 & 11.0 & 15.2 \\
18.8 & 15.2 & 13.6 & 36.2 & 16.2 \\
14.0 & 8.0 & 13.0 & 9.4 & 55.6
\end{array}\right)
\end{array}
\tag{B.1}
$$

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	17.557	6.230	1.000	0.568	0.098	-0.075	0.440
β	0.443	0.068	0.568	1.000	0.290	-0.055	0.212
snr	2.176	0.033	0.098	0.290	1.000	-0.425	-0.137
S	4.788	0.748	-0.075	-0.055	-0.425	1.000	0.132
α_1	0.430	0.109	0.440	0.212	-0.137	0.132	1.000

Table B.1: New mean, standard deviation and correlation for class A using the corrected kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	26.003	12.184	1.000	0.622	0.445	-0.447	0.144
β	0.401	0.076	0.622	1.000	0.473	-0.313	0.016
snr	2.180	0.055	0.445	0.473	1.000	-0.175	-0.323
S	3.590	1.329	-0.447	-0.313	-0.175	1.000	-0.464
α_1	0.326	0.168	0.144	0.016	-0.323	-0.464	1.000

Table B.2: New mean, standard deviation and correlation for class B using the corrected kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	29.708	14.246	1.000	0.453	-0.051	-0.187	0.459
β	0.478	0.092	0.453	1.000	-0.014	-0.076	0.474
snr	2.199	0.051	-0.051	-0.014	1.000	-0.479	0.122
S	4.205	1.107	-0.187	-0.076	-0.479	1.000	-0.424
α_1	0.473	0.181	0.459	0.474	0.122	-0.424	1.000

Table B.3: New mean, standard deviation and correlation for class C using the corrected kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	26.058	11.769	1.000	0.043	0.464	-0.289	0.091
β	0.430	0.108	0.043	1.000	0.092	-0.137	0.511
snr	2.198	0.048	0.464	0.092	1.000	-0.321	-0.030
S	4.131	1.187	-0.289	-0.137	-0.321	1.000	-0.386
α_1	0.461	0.221	0.091	0.511	-0.030	-0.386	1.000

Table B.4: New mean, standard deviation and correlation for class D using the corrected kernel based method.

x	μ_x	σ_x	$\rho_{\mu_A x}$	$\rho_{\beta x}$	$\rho_{snr x}$	$\rho_{S x}$	$\rho_{\alpha_1 x}$
μ_A	25.986	13.666	1.000	0.401	0.086	-0.371	0.336
β	0.458	0.107	0.401	1.000	0.168	-0.124	0.209
snr	2.219	0.034	0.086	0.168	1.000	-0.526	-0.282
S	4.038	1.263	-0.371	-0.124	-0.526	1.000	0.192
α_1	0.452	0.118	0.336	0.209	-0.282	0.192	1.000

Table B.5: New mean, standard deviation and correlation for class E using the corrected kernel based method.

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 81.4 & 3.8 & 4.2 & 1.2 & 9.4 \\ 4.6 & 71.2 & 4.4 & 9.8 & 10.0 \\ 16.6 & 17.4 & 37.8 & 11.6 & 16.6 \\ 16.2 & 15.0 & 16.4 & 38.8 & 13.6 \\ 14.8 & 8.0 & 14.8 & 7.2 & 55.2 \end{pmatrix} \end{array} \tag{B.2}$$

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 85.3 & 3.1 & 3.1 & 0.0 & 8.5 \\ 0.0 & 90.0 & 0.0 & 0.0 & 10.0 \\ 28.6 & 14.3 & 35.7 & 7.1 & 14.3 \\ 23.1 & 7.7 & 23.1 & 30.8 & 15.4 \\ 10.0 & 10.0 & 10.0 & 10.0 & 60.0 \end{pmatrix} \end{array} \tag{B.3}$$

If the "optimal" back-propagation network for the ABCDE-problem, which has been determined in section 5.4.7, is trained with the new training set, while classifying the validation set, figures B.1 and B.2 arise.

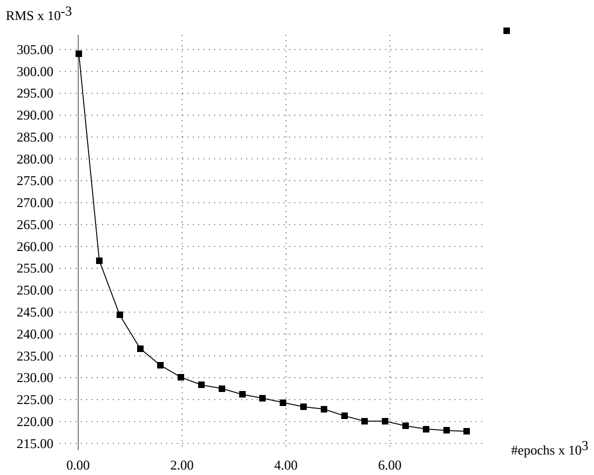


Figure B.1: Performance of back-propagation on the corrected ABCDE-training set.

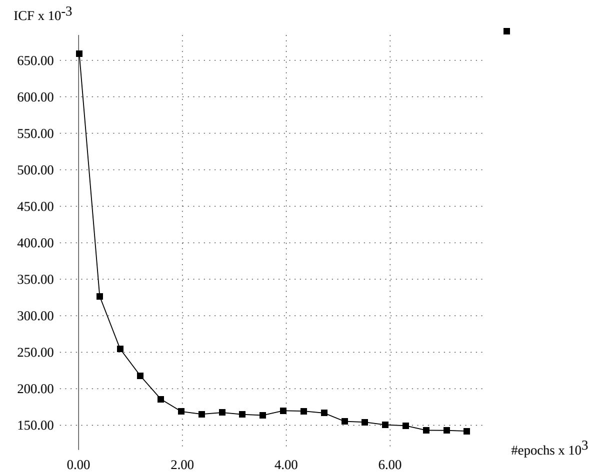


Figure B.2: Performance of back-propagation on the corrected ABCDE-validation set.

After 2324 epochs the performance on the validation set is optimal; the classification percentages that are achieved in this case can be found in matrix B.4.

$$
\begin{array}{c}
 \\ A \\ B \\ C \\ D \\ E
\end{array}
\begin{array}{c}
\begin{array}{ccccc} \quad A & \quad B & \quad C & \quad D & \quad E \end{array} \\
\begin{pmatrix}
81.4 & 2.3 & 3.1 & 5.4 & 7.8 \\
0.0 & 100.0 & 0.0 & 0.0 & 0.0 \\
14.3 & 0.0 & 85.7 & 0.0 & 0.0 \\
7.7 & 0.0 & 0.0 & 92.3 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 100.0
\end{pmatrix}
\end{array}
\tag{B.4}
$$

In figures B.3 and B.4 the performance of the "optimal" feature mapping network — defined in section 6.4.7 — on the new training and validation set can be found, when there is to be discriminated between five classes simultaneously.

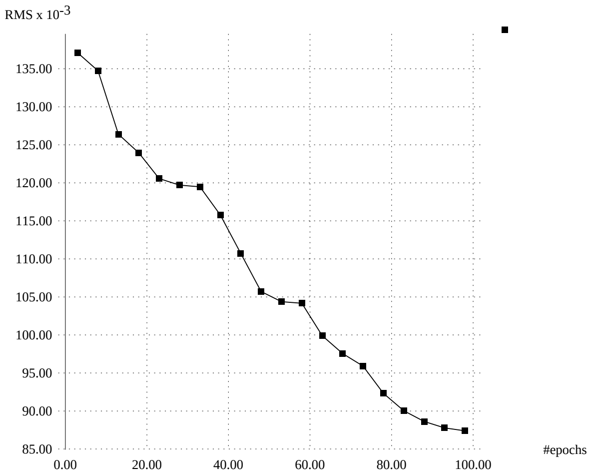


Figure B.3: Performance of feature mapping on the corrected ABCDE-training set.

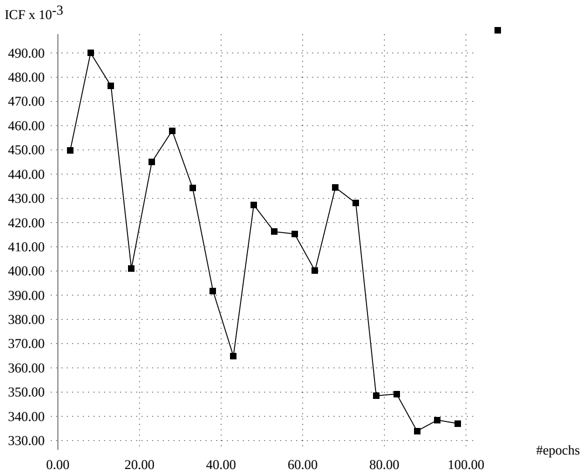


Figure B.4: Performance of feature mapping on the corrected ABCDE-validation set.

The performance on the validation set is optimal after 88 epochs; in matrix B.5 the classification percentages for this situation are presented.

$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} \quad A & \quad B & \quad C & \quad D & \quad E \end{array} \\
\begin{pmatrix}
67.4 & 9.3 & 7.8 & 3.9 & 11.6 \\
0.0 & 80.0 & 0.0 & 20.0 & 0.0 \\
21.4 & 14.3 & 57.1 & 7.1 & 0.0 \\
15.4 & 15.4 & 7.7 & 61.5 & 0.0 \\
0.0 & 10.0 & 10.0 & 0.0 & 80.0
\end{pmatrix}
\end{array}
\tag{B.5}
$$

In chapter 7 it was mentioned that the conclusions can be divided in two parts: one part dealing with the use of ultrasonic tissue characterization on our specific application, while the other part deals with the comparison of the methods that have been investigated.

In this section it can be found that the percentages of correct classification are much higher for the correctly generated data than before, resulting in an increased practical use of the application. However, the results on the validation set (matrix B.3) are very similar to the results of discriminant analysis on the original data using resubstitution (matrix 4.42), which suggests that the generated data may be too dependent on the original data. This is something that has to be investigated in the future.

Comparing the matrices B.3, B.4 and B.5 with table 7.1, it can be seen that the order in which the investigated methods are able to discriminate between five classes simultaneously is still the same: back-propagation is the best classifier, followed at some distance by feature mapping, which at its turn is closely followed by discriminant analysis.

Summarizing the influence of the error in the data generation program on the conclusions of this thesis, it can be said that the practical use of the application is increased when using the correctly generated data, but the order in which the different methods can best be used as classifier stays the same.