Automated Design of Neural Network Architecture for Classification

Ph.D. Thesis

by

Jan Depenau

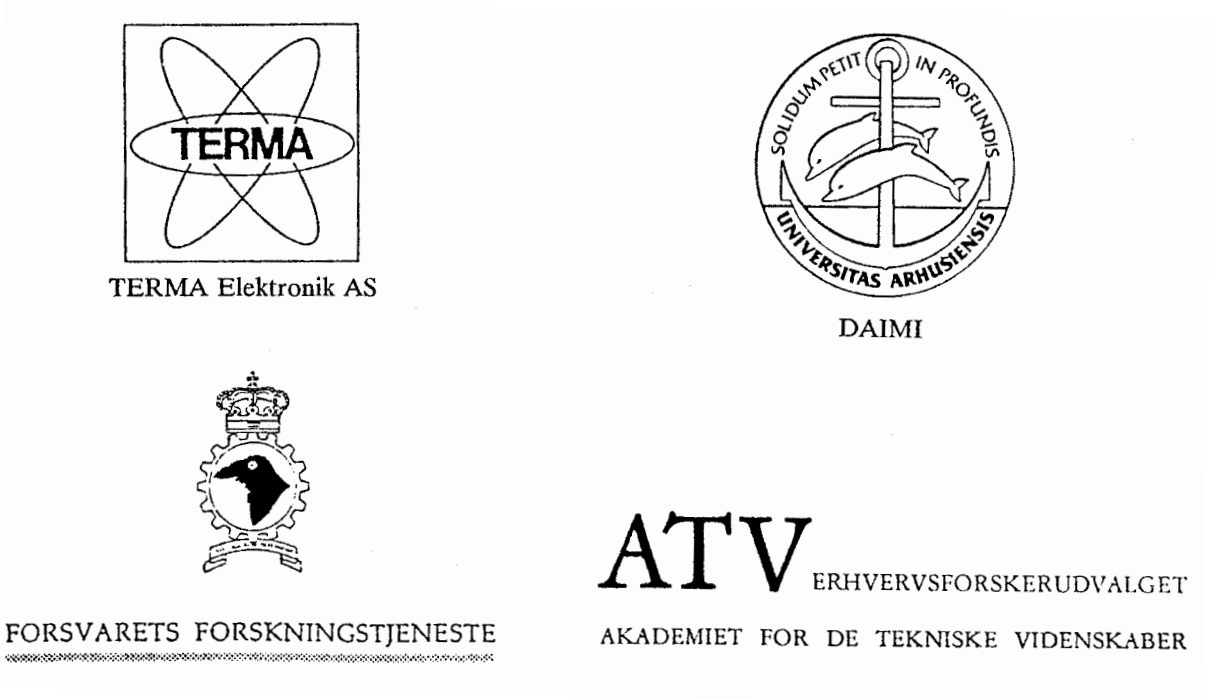
TERMA Elektronik AS, Hovmarken 4, DK-8520 Lystrup

and

DAIMI, Computer Science Department, Aarhus University,

Ny Munkegade, Bldg. 540, DK-8000 Aarhus C

August 1995



Danish Academy of Technical Sciences Industrial Research Education Ph.D. Thesis EF-448

Automated Design of Neural Network Architecture for Classification

Ph.D. Thesis
by

Jan Depenau

August 1995

Contents

Preface

The present thesis is submitted to the Faculty of Natural Sciences, Aarhus University, Denmark to fulfil the requirements for the Ph.D. degree under the Industrial Research Education Programme.

Besides serving the above-mentioned purpose, it is also my hope that colleagues from the industry and graduate students who are interested in neural networks will find it useful. For those who are not very familiar with neural networks or beginners in this field a glossary, included as appendix A, might be helpful. A word included in the glossary will be marked with a heart $^{\heartsuit}$ the first crucial time it occurs in the text. The references in the bibliography are organised in groups according to contents. The list includes not only references made in this thesis, but also high class references that provide an additional view of things are included.

The thesis constitutes a major part of the research work that was carried out during an Industrial Research project from March 1993 to August 1995. The work in this thesis deals with finding a good architecture of a neural network classifier. The focus is on methods to improve the performance of existing architectures (i.e. architectures that are initialised by a good academic guess) and automatically building neural networks.

The thesis is partly built on the 4 following papers and 2 technical reports that have already been published, and partly on material that gives an introduction to the papers and/or puts them in perspective. The thesis, however, can be read as a complete and independent text. The papers and technical reports are:

Aspects of Generalization and Pruning, Proceedings from the World Congress on Neural Networks, Vol 3, pp. 464-469, San Diego 94. J. Depenau and M. Møller.

Soft-Monotonic Error Functions, Proceedings from the World Congress on Neural Networks, Vol 3, pp. 444-449, San Diego 94. M. Møller and J. Depenau.

Evaluation of the Cascade Correlation Algorithm, Technical Report. 23 pages.
J. Depenau.

A Global-Local Learning Algorithm, Proceedings from the World Congress on Neural Networks, Vol 1, pp. 587-590, Washington 95. J. Depenau.

The MS-σ Error Function, Proceedings from the World Congress on Neural Networks, Vol 1, pp. 614-617, Washington 95. J. Depenau and M. Møller.

Experiments with Different Neural Network Architectures for Classification of Ice Type Concentration from ERS-1 SAR Images Technical Report. 22 pages.
J. Depenau.

The papers and one of the technical reports are included in appendices C - G while the last report is included in chapter 7.

The Industrial Research project was carried through in a cooperation between TERMA Elektronik AS, Lystrup, Denmark, DAIMI, Computer Science Department, Aarhus University, Aarhus C, Denmark and the Danish Defence Research Establishment, Copenhagen, Denmark. The administration of the Industrial Research Education Programme is carried out by the Danish Academy of Technical Sciences. The project was partly financed by ATV and partly by TERMA Elektronik AS.

Aarhus, August 1995

Jan Depenau

Abstract

During the last ten years neural networks have shown their worth, particularly in areas such as classification. The success of a neural network approach is deeply dependent on finding the right network architecture. The architecture of a neural network determines the number of neurons in the network and the topology of the connections within the network. The emphasis of this thesis is on automatic generation of network architecture.

The background for this thesis, the Industrial Research Project, is described in chapter 1. The description builds up around the educational plan for the project and includes problem definition, goals and possible solution. The educational plan in full length can be found in appendix B.

In chapter 2 an introduction to the Multi-Layer feed-forward neural network is given. It starts with a short description of the theoretical foundation, to establish the notation and basic definitions used in this thesis.

The next two chapters deal with one of the most essential properties for neural networks as well as for any learning machine: the ability to learn from examples. In chapter 3 training and measuring of performance are discussed. A review of some of the most used training algorithms and error functions is given. Further three new error functions, two Soft-monotonic error functions and the MS-σ error function are introduced. Both ideas have been published in conference papers, which are found in appendices D and G.

The objective of training a network is to make it able to find the underlying truth: i.e. to generalise. In order to understand the meaning of generalisation, how to estimate it and point out some problems with measuring, various definitions are provided in chapter 4. The focus is on the framework of *Empirical Risk Minimisation (ERM)* and *Probably Approximately Correct (PAC) learning theory*. Both the *ERM* and the *PAC* theory are based on the measurement of a machine's complexity, in terms of the *growth function* and the *VC-dimension*, which in general is unknown. An exception is the linear classifier and it is shown that many of the self-constructing algorithms developed during the last few years have the same *VC-dimension* as a linear classifier.

The various generalisation theories seem to agree that the model/machine/neural network that is able to learn the training data in a satisfactory way using the smallest number of parameters, is the model/machine/neural network with the largest probability of a good generalisation ability. (This philosophy is often called Ockham's Razor). This means e.g. that a network with many parameters that gives a low learning error should not be preferred to a network with few parameters giving a slightly larger, but still satisfactory, learning error. From a neural network point of view several ways have been proposed to

achieve this goal. In general the methods can be split into two groups: one where the methods try to reduce the number of units or weights in a well working network, and another where the methods focus on successively building a network.

In chapter 5 the first approach is explored. Various methods are described and some of the most popular are tested. It has often been discussed which method is the best. On the basis of the theory from chapter 4 and experiments it is argued that none of the methods is better than the others. This work has been published in conference papers, which are placed in appendix C. The construction ideas are considered in chapter 6. A review of well established methods is given. Among these the *Cascade-Correlation Algorithm (CCA)* has been studied the most. In appendix E a detailed description and analysis of the *CCA* is provided. New ideas of combining units with different types of transfer functions like radial basis functions and sigmoid or threshold functions led to the development of a new construction algorithm for classification. The algorithm called *GLOCAL* is fully described in chapter 6 while a shorter version published in conference proceedings can be found in appendix F.

Several of the methods described in the previous chapters were used on real life data from a *Synthetic Aperture Radar (SAR)* in order to classify different ice types. In chapter 7 a description of the ice problem, previous work in that area, and results from experiments made in a preliminary investigation are provided. The chapter is identical to the technical report *Experiments with Different Neural Network Architectures for Classification of Ice Type Concentration from ERS-1 SAR Images* released in June 1995.

The overall conclusion of the work performed in connection with the research and documented in this thesis is that there are several possible methods or strategies for automatically building neural networks that are able to solve large problems like the SAR task. This conclusion is supported in chapter 8 where the work described in the previous chapters is put in perspective in relation to the three educational subgoals and the new things to which this thesis has contributed are highlighted. Suggestions on future works and remarks from the author are also included.

Acknowledgements

A large number of people have made this research possible. I would like to express my gratitude to all of them.

First I would like to thank TERMA Elektronik AS for giving me the chance to do this work and for their support throughout the project. I would like to thank all members of the project management committee: **Birger Nielsen** and **Hans Peter Kyk**, TERMA Elektronik AS, **Brian Mayoh**, Computer Science Department, Aarhus University, **Gert Hvedstrup Jensen**, The Danish Defence Research Establishment, and **Ove Skovgaard** representing the Danish Academy of Technical Sciences, for their support and many useful discussions during the project.

I would like to offer my best thanks to **Professor Brian Mayoh** for being a very inspiring advisor, always positive and encouraging.

Many hours have been spent creating new ideas, discussing all kinds of technical issues and making good and foolish experiments together with my friend and collaborator **Martin Møller** for which I am very grateful.

I would also like to express my gratitude to **Morten Buhr** for helping with experiments and valuable proofreading, and to **Henning Skriver** for making the SAR-data available and supplying me with his results and data.

Last, but not least, I am indebted to my family for their support and understanding.

The aid of others was invaluable, but I alone am responsible for the opinions, technical details, and faults of this dissertation.

Résumé in Danish

Inden for de sidste ti år har neurale netværk vist sig at være anvendelige inden for en lang række områder og i særdeleshed i forbindelse med klassifikationsopgaver. En af de vigtigste faktorer for, at anvendelsen af et neuralt netværk bliver en succes, er, at den rigtige netvæksarkitektur findes. Et neuralt netværks arkitektur bestemmer antallet af beregningsenheder (neuroner) i netværket og topologien af forbindelserne i netværket. I denne afhandling fokuseres der på metoder til automatisk generering af netværksarkitekturer.

Baggrunden for denne afhandling er Erhvervsforskerprojektet EF-448, som er beskrevet i kapitel 1. Beskrivelsen er bygget op omkring uddannelsesplanen for projektet og indeholder bl.a. problemstilling, mål for projekt og uddannelse samt mulig løsning. Selve uddannelsesplanen findes i sin fulde længde i appendix B.

Kapitel 2 indeholder en præsentation af et af de mest populære og anvendte neurale netværkstyper kaldet multi-layer feed-forward netværk. Der startes med en kort beskrivelse af det teoretiske grundlag, som danner baggrund for en introduktion til notationen og de grundlæggende definitioner, der er anvendt i denne afhandling.

De næste to kapitler omhandler en meget vigtig egenskab for såvel neurale netværk som for enhver anden "*Indlæringsmaskine*": evnen til at lære på basis af eksempler. I kapitel 3 beskrives, hvorledes et neuralt netværk trænes, og hvorledes dets performance kan måles v.h.a. fejlfunktioner. Der gives en oversigt over de mest anvendte træningsalgoritmer og fejlfunktioner. Desuden introduceres tre nye fejlfunktioner, to *soft-monotonic* fejlfunktioner samt en *MS-σ* fejlfunktionen. Disse funktioner er også beskrevet i konferenceartikler, se evt. appendix D og G.

Formålet med at træne et netværk er at sætte det i stand til at lære den bagvedliggende sandhed, der er gemt i dataene. I denne forbindelse tales der om netværkets evne til at generalisere. For at forstå betydningen af generalisering, hvordan det måles og påpege visse problemer i forbindelse med måling, gives der en række definitioner i kapitel 4. Der fokuseres på to hovedområder: *Empirical Risk Minimisation (ERM)* og *Probably Approximately Correct (PAC) learning theory*. Både ERM og PAC-teorien er baseret på et mål af en maskines kompleksitet, udtrykt ved en *growth funtion* og en *VC-dimension*, som normalt er ukendt. En undtagelse er en lineær klassifikator, og det vises, at mange af de selvkonstruerende algoritmer, der er blevet udviklet i de senere år, har den samme VC-dimension som en lineær klassifikator.

De forskellige generaliseringsteorier synes at være enige om, at det neurale netværk, som er i stand til at indlære træningsdata på en tilfredsstillende måde ved hjælp af det

mindste antal parametre, er det neurale netværk, som har størst sandsynlighed for at opnå en god generaliseringsevne. (Denne filosofi benævnes ofte Ockham's Razor.) Dette betyder f.eks., at et netværk med mange parametre, som giver en lav indlæringsfejl, ikke bør foretrækkes frem for et netværk med få parametre, selvom dette giver en lidt større, men stadig acceptabel, indlæringsfejl. I forbindelse med neurale netværk er der blevet foreslået mange måder til at opnå dette. Generelt kan metoderne deles op i to grupper: en gruppe af metoder, som prøver på at reducere antallet af beregningsenheder eller vægte i et velfungerende netværk, og en anden gruppe af metoder, som fokuserer på successiv opbygning af netværk.

I kapitel 5 udforskes den første fremgangsmåde. Forskellige metoder beskrives, og nogle af de mest populære testes. Det er ofte blevet diskuteret, hvilken metode der er den bedste. På basis af teorien i kapitel 4 og de udførte eksperimenter argumenteres der for, at ingen af metoderne er bedre end de andre. Dette arbejde er blevet offentliggjort i en konferenceartikel, som findes i appendix C. Ideerne om konstruktion (successiv opbygning) er taget under overvejelse i kapitel 6. Der gives en oversigt over gennemprøvede metoder, bl.a. Cascade-Correlation Algorithm (CCA), som er blevet mest afprøvet. Appendix E indeholder en detaljeret beskrivelse og analyse af CCA. Nye ideer, der kombinerer brugen af beregningsenheder, der anvender henholdsvis radial basis-funktioner og sigmoid-funktioner, førte til udvikling af en ny konstruktionsalgoritme til klassificering. Algoritmen, der kaldes GLOCAL, er beskrevet fuldt ud i kapitel 6, mens den er kortere beskrevet i konferenceartiklen i appendix F.

Flere af de metoder, der er beskrevet i de foregående kapitler, blev anvendt i forbindelse med klassificering af forskellige istyper på baggrund af data fra en Synthetic Aperture Radar (SAR). Kapitel 7 indeholder en beskrivelse af isproblematikken, tidligere arbejde udført inden for dette område og resultater fra eksperimenter foretaget i forbindelse med en foreløbig undersøgelse. Dette kapitel er identisk med den tekniske rapport Experiments with Different Neural Network Architectures for Classification of Ice Type Concentration from ERS-1 SAR Images udgivet i juni 1995.

Den overordnede konklusion på det arbejde, der er udført i forbindelse med forskningen og dokumenteret i denne afhandling, er, at der er flere mulige metoder eller strategier til automatisk opbygning af neurale netværk, som er i stand til at løse store opgaver som f.eks. SAR-opgaven. Denne konklusion underbygges i kapitel 8, hvor arbejdet beskrevet i de foregående kapitler sættes i perspektiv i forhold til de tre uddannelsesmæssige delmål, og de nye tiltag, som denne afhandling har bidraget til, fremhæves. Desuden indeholder dette kapitel også forslag til fremtidigt arbejde samt forfatterens kommentarer.

Jan Depenau

DAIMI, Aarhus Universitet
August 1995

Chapter 1

Introduction

This chapter gives a general presentation of the Industrial Research Education which provides the background for this thesis. A substantial part of this chapter is a short version of the Educational Plan formulated at the start of the project and reviewed in November 94. The full Educational Plan can be found in appendix B.

1.1 Project Background

TERMA Elektronik AS designs, develops and produces a broad spectrum of eletronic equipment from small RADAR systems to large complex Command, Control, Communication and Information systems (**C3I**). TERMA expects that in future applications of these areas there will be a need for neural network methods for recognition and classification.

The main goal of a neural network classifier is to get a system that is able to classify unknown data correctly, i.e. a system with a good generalisation ability. At present it is known that there is a relationship between the capacity of a network and its generalisation ability. Further it is also known that the capacity of a feed-forward network is somehow related to the number of units and weights in a network, and thereby the network's architecture. [1] The mathematical underpinnings on these issues are rather weak, but various strong attempts are being made to attack these questions. A more practical approach to overcome this problem has also been proposed. Various more or less mathematically based methods have been developed.

This project deals with the problem of how to find a good architecture of a neural network classifier. The focus is on methods to improve the performance of existing architectures (i.e. architectures that are initialised by a good academic guess) and methods for automatically building neural networks.

[1] The architecture of a neural network determines the number of neurons in the network and the topology of the connections within the network.

1.1.1 Project's Educational Goals

The project's educational goals were split into three categories:

a. Research level

Establish a high level of knowledge of methods and techniques for building neural network classifiers. The success of a neural network is strongly dependent on finding the right network architecture. The emphasis of this research will be on automatic generation of network architecture and methods to improve the performance of existing architectures. Various promising methods have been proposed and the area is currently receiving increased attention. The existing methods can be categorised into two major areas. Some methods focus on successively building a network while others try to reduce the number of units or weights in a well working network. The optimal solution is expected to be a strategy that combines these two approaches.

b. Development level

Methods and algorithms for automatically generating neural network architectures or optimising existing neural networks are to be developed and implemented and then used to build an artificial neural network which is able to perform classification of various radar sources. The neural network must accept preprocessed data from a Synthetic Aperture Radar (SAR) and perform an analysis which establishes a classification of different objects (ice types).

c. Application level

It is expected that the developed algorithms and methods will have such a general structure that they will be useful for a broad spectrum of other applications. Examples of such applications go from Classification of Radar Targets, Electronic Data Interchange (EDI), Data Fusion, to Analysis of formatted messages to or from a C3I system.

1.2 Project Description and Project Phases

The overall purpose of the project "Automated Design of Neural Network Architecture for Classification" is to investigate the possibility of finding a method or strategy for automatically building neural networks so the network will be able to classify objects in radar signals.

More specifically the project includes research, development, and implementation of methods that build neural networks (methods and algorithms), performing classification of "radar" signals.

Although an enormous effort in the area of research and development of neural networks has been carried out, there is yet no simple or unambiguous answer to the question of how to choose or build a suitable neural network for a specific problem. Over the last decade more than 50 different neural network types have been suggested. To go into a detailed study of each of these network types would be a fatiguing task and take a long time.

Fortunately, work has been carried out within the area to outline the difference between various types. It seems that several generic neural network structures are useful in connection with the classification problems in this project.

The network of choice for classification (and many other applications) is a multi-layer feed-forward network. Although the multi-layer feed-forward network seems to be a reasonable choice of network, other types of network like Cascade networks and Self-Organising networks will also be considered.

These networks give an idea of the way to look for solutions but one still needs to decide how to build the network. There are several ways of approaching the solution to this problem from a practical point of view. The fastest way to build a network that solves a specific problem in accordance with the desired performance, is to use a well-known simulator software and experiment. There are, however, two major drawbacks to the simulator approach. There is no guarantee that the solution is optimal as regards architecture and solution. There are many examples that a well working net has shown a better performance when some of its connections/neurons are removed. Other examples show that learning rate and learning time are increased when an extra connection is added. It will of course be possible to find an optimal solution with a simulator, by adding or deleting connections/neurons, but it may take a very long time and be very laborious. And the ultimate network will only be valid for the specific problem, which means that one will not be able to apply the results to other problems or even the same problem if new inputs were to be added.

Another approach without these drawbacks is to use a method that not only optimises the weights for a given architecture, but also optimises the architecture itself. For this project there are several indications that this approach is the most reasonable:

- It offers a way to address problems that are general for neural network building problems. This means that one can start working without having data from the problem domain, and instead use other data that relate to the problem domain. This is not the case when one is working with a simulator in order to find a solution.
- It can be easily adapted to radar classification problems.
- It will be useful for other applications.

1.3 Strategy

The overall philosophy of the EF448 project is: first to analyse state-of-the-art methods and develop new methods for dealing with the general problem of how to find the architecture of a neural network. The focus will be on methods for automatically building neural networks and especially methods that not only optimise the weights for a given architecture, but also optimise the architecture itself. Second to use the results from this research to develop a method that automatically builds a neural network being able to perform a classification of radar-like signals.

The project has been split into 4 phases:

- **1st Phase: Problem analysis (6 months).** A further introduction to the issue, including a study of literature, which besides usual technical literature also covers military areas. On the basis of this analysis, scope and constraints of the project are determined. Points and criteria to be investigated and studied during the following investigation phase are defined.

- **2nd Phase: Investigation of methods and algorithms (6 months).** A further investigation, including experiments and simulations, of different methods and algorithms which automatically build neural networks. This covers methods, available at present, state-of-the-art methods implemented or under implementation and own proposals. Points and criteria to be elaborated in the following detailed study phase are defined.

- **3rd Phase: Detailed study (9 months).** A detailed study of the most promising methods and algorithms for automatically building neural networks. Implementation of a prototype which makes it possible to build a neural network that will be able to perform a classification.

- **4th Phase: Implementation and test of the neural network (9 months).** This phase includes: collection and analysis of data, including construction of a programme for loading these data and implementation of a neural network which can perform a classification of SAR radar signals. In parallel with the implementation, tests are also carried out. At the end of this phase a test and performance evaluation of a prototype of the final neural network is performed. About 2 months of the effort in this phase is concentrated on the composition of the Ph.D. thesis.

At the start of the project the original task was expected to be classification of formatted messages. It turned out that the data would not be accessible within the period of this project. This just confirms that the chosen strategy was right because only small adjustments were needed in order to keep the project on the right track and minimise the waste of work.

1.4 Related Reports and Papers

A report on progress has been released every three months. These reports include: status of the project, future work, evaluation and description of courses and conferences, proposal for the Neural Network Project and exercises for last year students and finally technical material. A Newsletter was released at the start of the project. 2 technical reports were sent to colleagues and fellow researchers, 4 papers were published in conference proceedings.

Chapter 2

Neural Networks

The aim of this chapter is to give a brief introduction to neural networks and provide the reader with a technical foundation and with the terminology used in this thesis. Terms like feed-forward network, perceptron, multi-layer perceptron, radial basis function network and classification are presented.

2.1 Overview - Introduction

Artificial neural networks or simply neural networks are often seen as implying an intention to duplicate the information processing of the nervous system. Early work in the field was indeed inspired by biological studies of the brain function, and [McCulloch and Pitts 43] and [Hebb 49] described systems of neurons and learning rules derived from biological models. Today most neural networks$^{\heartsuit}$ bear only a passing resemblance to natural brains or nervous systems. The relationship between a modern neural network and the brain of an animal lies in the idea of performing computations by using the parallel interaction of a very large number of simple entities.

Neural networks have been used in connection with many different applications. The tasks for which they are used are generally problems of pattern recognition/classification or function approximation. Typically, a network will be asked to classify an input pattern as belonging to one of a number of different possible classes, or to produce an output value as a (previously) unknown function of one or more input values. The crucial feature of neural networks is their ability to learn how to make the desired mapping from inputs to outputs without explicitly having to be told the rules for doing so. Instead, they adjust their internal connections based on a number of examples of the required mapping, and are then used to generalise from the given examples to others that they have not previously seen. Thus they are used for capturing patterns in sets of data, and use these captured patterns to perform the required computations.

There are many different types of neural networks, each with its own advantages and drawbacks depending on the task. The network of choice for classification (and many other applications) is a **feed-forward network**, including a **simple perceptron** and its extension the **multi-layer perceptron**. Feed-forward networks are characterised by the fact that the information starting from the input is only allowed to pass forward in the network to the output, no feedback is allowed.

In this thesis considerations will be restricted to various types of feed-forward networks. For further introduction to history, biological relations and applications, a group of references can be found in the bibliography under the heading *"GENERAL DESCRIPTION OF NEURAL NETWORKS"*.

2.2 Basic Definitions and Notation

All neural networks consist of a number of highly interconnected computing **units**$^{\heartsuit}$, also called neurons$^{\heartsuit}$ or nodes. Each unit receives inputs from other units in the network, or from the outside world, and calculates an output based on these inputs. Associated with each **connection** (sometimes called a synapse) between the units is a **weight**$^{\heartsuit}$. The way in which the units are connected, and how they communicate with the outside world, exerts a vital influence on the network's properties.

For the feed-forward network type, units are arranged into layers$^{\heartsuit}$. It consists of L processing levels, where the first level $l = 0$ is the **input layer** and the last level $l = L$ is the **output layer**. All communication with the outside world is done through these two layers. Usually intermediate layers of units which cannot be reached from the outside world are present. They are called **hidden layers** and the units within them are called **hidden units**. The hidden layer closest to the input is called the first hidden layer, the next layer second hidden layer and so on. The last hidden layer is therefore the one closest to the output layer. If it does not have any hidden layers, the network is called a **simple perceptron** otherwise it is called a **multi-layer perceptron** or **X-layer perceptron** where X is the number of hidden layers added by 1, representing the output layer, the input is usually not counted. In a feed-forward network the information passes from a lower layer forward to a higher. This means that units from a layer may only be connected to units in higher layers, no feedback or intraconnections are allowed. A typical multi-layer feed-forward network$^{\heartsuit}$ with the notational conventions is shown in figure 2.1.

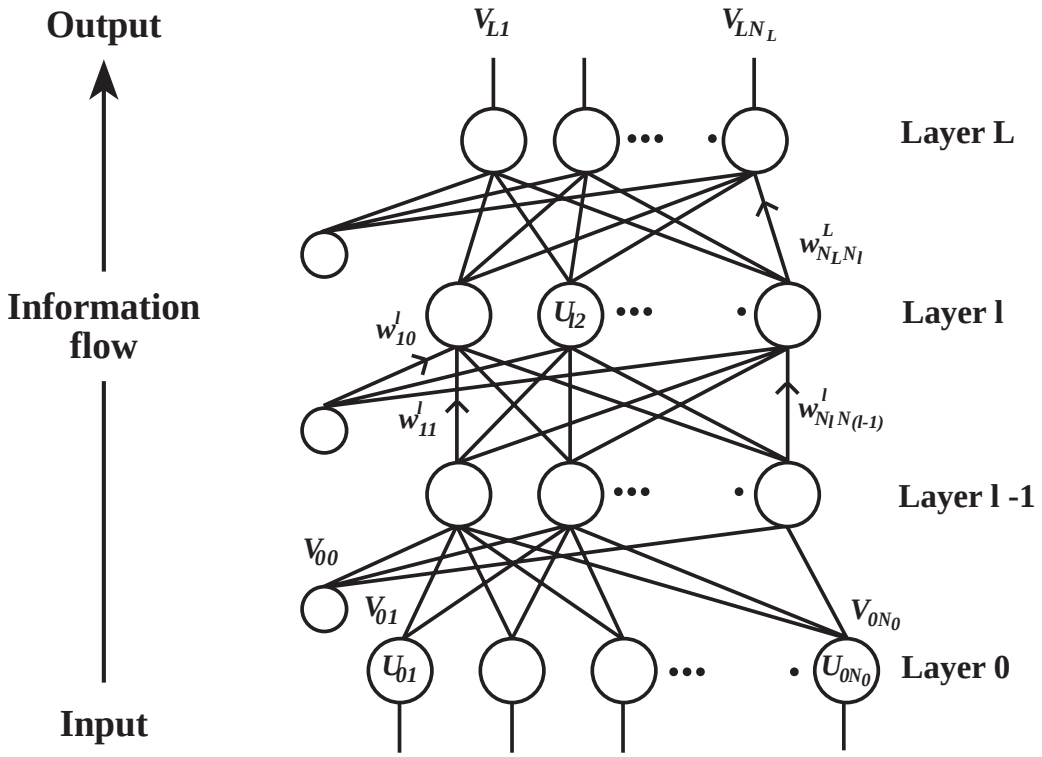


Figure 2.1: Example of a typical Multi-layer feed-forward network, showing the notation used in this thesis.

The **network architecture**$^\heartsuit$ is specified by the number of units per layer N_l where $0 \leq l \leq L$ and by the **topology** of the connections within the network. The parameters of the network are the weights w^l_{ij} of the connection between units in layer l and units in the previous layer $l-1$ and the biases w^l_{i0} to unit i in layer l, where $1 \leq i \leq N_l$, $1 \leq j \leq N_{l-1}$ and $0 \leq l \leq L$. **Throughout the thesis the biases will from time to time be omitted in the illustrations, although the effect of the biases is counted**.

The network works as follows: An input pattern is presented to the network via the input layer which is just a set of data latches that keep the value of the input fixed. The processing then passes through the intervening layers of hidden units before reaching the output. The entire network can be regarded as an input-output mapping function, that given an input $\mathbf{V}_0$ will produce an output $\mathbf{V}_L$, thus $\mathbf{V}_L = f_{\mathbf{w}}(\mathbf{V}_0)$. The subscript of $\mathbf{w}$ corresponds to the function's dependence on the weights.

The processing of each level is determined by a dynamical relation between the output **values** V_{l-1} of units in layer $(l-1)$ and the units in layer l. The **activity**$^\heartsuit$ inside the $i'th$ unit in layer l, denoted U_{li}, determines the state V_{li} via a **transfer function**$^\heartsuit$ so that $V_{li} = g(U_{li})$. U_{li} is also used to denote the current unit while it unambiguously determines the unit's placement in the network.

The most significant differences between the networks considered in this thesis are the choice of transfer function and the calculation of activity for each unit.

2.3 Multi-Layer Perceptron

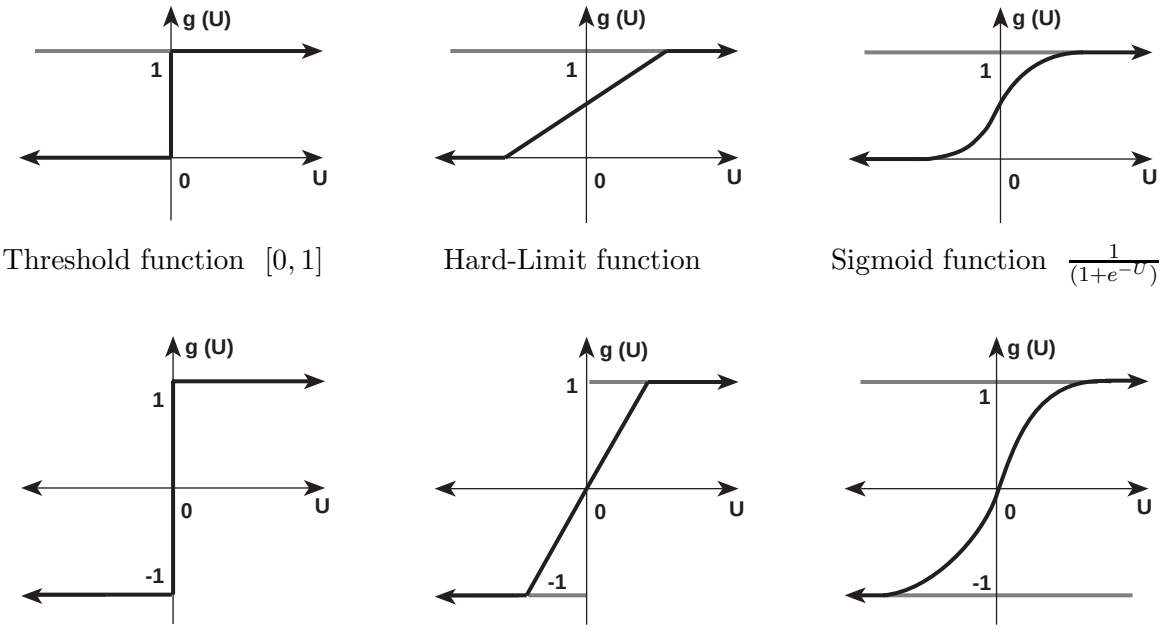
The architecture of the MLP follows the definition of the feed-forward network described in the previous paragraph without any restriction. The activity and output value from each unit in an MLP are determined by the following rule:

$$\text{Activity: } U_{li}^p = \sum_{j=1}^{N} w_{ij}^l V_{(l-1)j}^p - \theta_{li} \tag{2.1}$$

$$\text{Output: } V_{li}^p = g(U_{li}^p), \tag{2.2}$$

where p is the pattern number, w_{ij}^l is the weight from the $j'th$ unit in layer $l-1$ to the $i'th$ unit in layer l, θ is the threshold, and g is a transfer function. Sometimes it is convenient to omit the threshold term, which can be done by defining a fictive value $V_{(l-1)0} = 1$ and setting $w_{(l-1)0} = \theta_{li}$ as shown in figure 2.1. The index j is then started from 0 in equation (2.1).

A number of commonly used transfer functions are shown in figure 2.2.



Threshold function $[0, 1]$ | Hard-Limit function | Sigmoid function $\frac{1}{(1+e^{-U})}$

Threshold function $[-1, 1]$ | Hard-Limit function | Sigmoid function $Tanh(U)$

Figure 2.2: Example of commonly used transfer functions in an MLP.

The standard choice is $g(U_{li}^p) = \frac{1}{(1+e^{-(U_{li}^p)})}$ which confines all state variables to the $[0, 1]$ interval or $g(U_{li}^p) = Tanh(U_{li}^p)$ which confines all state variables to the $[-1, 1]$ interval. $Tanh(U_{li}^p)$ is often the best choice, a further explanation is given in [Le Cun 93a]. Both sigmoid and threshold functions [1] with an output interval between $[-1, 1]$ will be used as transfer functions in this thesis.

[1] The threshold function is sometimes called a *step function* if the output interval is between $[0, 1]$ and also a *sign function* if the output interval is between $[-1, 1]$.

2.4 Radial Basis Functions

The architecture of the MLP with radial basis function (RBF) units, also called an RBF network$^{\heartsuit}$, follows the definition of the feed-forward network described in a previous paragraph, but it is restricted to only containing one hidden layer of units whose output depends on a distance between the centre of an arbitrary transfer function and a given pattern. The output layer will commonly consist of standard linearly weighted units with a linear transfer function.

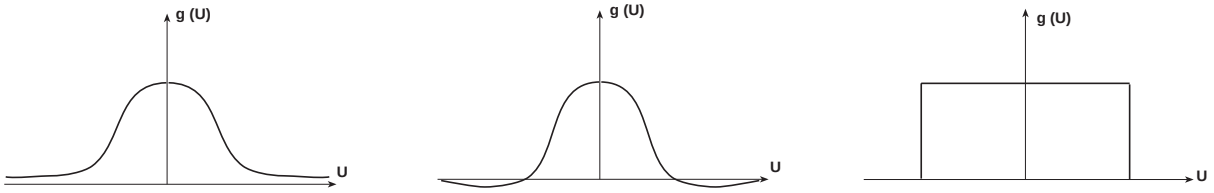
The activity and output value from each hidden unit in an RBF network are determined by the following rule:

$$\text{Activity: } U_{lj} = \| \mathbf{V}_{(l-1)} - \mathbf{C}_j \| \tag{2.3}$$

$$\text{Output: } V_{lj} = \Phi(U_{lj}) \tag{2.4}$$

where $\| \cdot \|$ is a norm on R^{N_0}, $\mathbf{V}_{l-1}$ is the input vector, $\mathbf{C}_j = \{c_{j1}, c_{j2}, \ldots, c_{jN_0}\}$, is the centre of the $j'th$ RBF, and Φ is typically a pulse function or a Gaussian function, performing a mapping from $R^{N_0} \rightarrow R_0^+$.

The use of such units is similar to the technique of radial basis functions for function approximation [Broomhead and Lowe 88]. This is an established technique, in which a wide variety of functions has been tried, for example Gaussians, Mexican hat and pulse functions, see figure 2.3.



Gaussian function | Mexican hat function | Pulse function

Figure 2.3: Example of commonly used transfer functions in RBF networks.

In this thesis Gaussians and pulse functions will be used. A detailed description of RBF networks can be found in the references under the heading *"RADIAL BASIS FUNCTION NETWORKS"* in the bibliography.

2.5 Classification

The problem of classification is basically a problem of partitioning a number of patterns (data) into classes $\mathcal{C} = \{c_1, c_2, \ldots, c_M\}$, where M is the number of classes. The problem considered in this thesis is how to **build** and make a neural network **learn** to classify an $N_0 - dimensional$ input into M classes, given P patterns. The input pattern is a number of features$^{\heartsuit}$ (might be raw data) represented by a vector of real values while the output is a vector with a dimension M with only one element turned on $(+1)$ at a time and the rest off (-1). This means that the network will perform a mapping from $R^{N_0} \rightarrow B^M$ where $B = \{-1, +1\}$.

The classification performed by a neural network will depend on the type of the transfer function used by the hidden units. If the hidden units employ threshold or sigmoid transfer functions, the network will become a **global** classifier. The reason is that each unit can be considered as a mechanism that splits the input space into two parts by a hyperplane. If the classification is done by an RBF network, the classifier will be **local**, because the unit's output depends on a distance between the centre of the transfer function and a given pattern. The units can thus be considered to be hyper-ellipsoidal.

In general a neural network can be described as a function that is able to perform a mapping from input to output so that the function makes an approximation of the true function. The question of which approximation a neural network can implement has often been asked, and the answer is that both MLP and RBF networks can approximate any function, see [Hornik *et al.* 89], [Leshno *et al.* 93] and [Hartman *et al.* 90], while a simple perceptron can solve only linearly separable$^{\heartsuit}$ classification problems [Minsky and Papert 69].

2.6 Networks Used in this Thesis

Besides the three networks mentioned a number of network types will be used in this thesis. In figure 2.4 there is an illustration of these networks and how they perform a classification task in two dimensions.

Although they might look different and are trained differently, they are all feed-forward networks. A further geometric description and interpretation of some of these networks and other types can be found in [Wieland and Leighton 87], [Lippmann 87], [Huang and Lippmann 87], [DARPA 88] (chapter 6), [Gibson and Cowan 90] and [Huang and Huang 91].

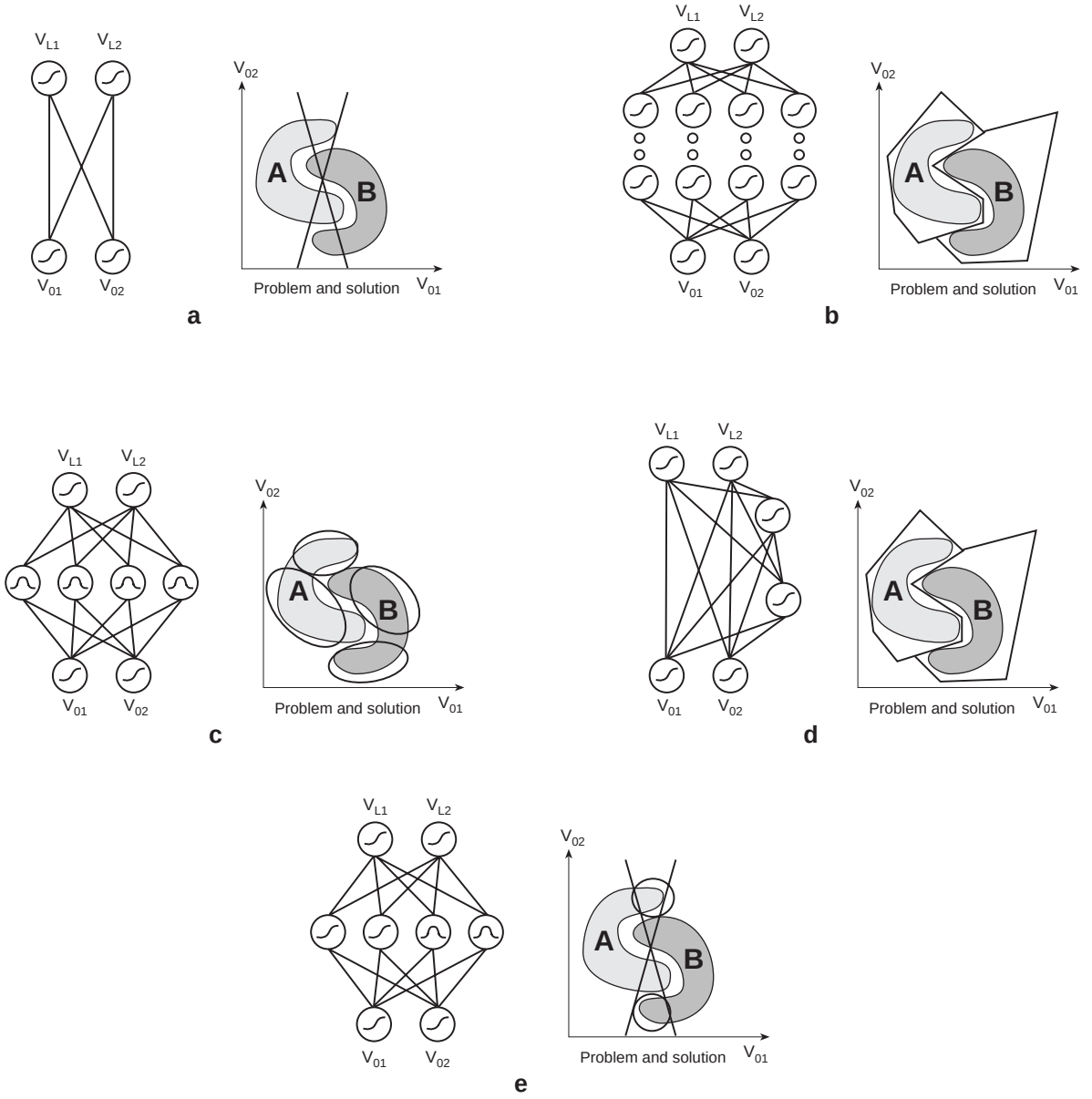


Figure 2.4: Example of networks used in the thesis and how they perform a classification task: a) Simple perceptron, b) Multi-layer perceptron, c) Radial Basis Function Network, d) Cascade-Correlation, and e) GLOCAL.

Chapter 3

Learning

Training is the process where the network's weights are adjusted in order to establish the desired input-output function. Training is one example of "machine learning", for a survey of other machine learning techniques see [Kocabas 91]. This chapter includes a description of different methods/techniques used in connection with learning for the three feed-forward neural networks: simple perceptron, RBF network and MLP. A short review of the most popular learning algorithms based on the gradient descent methods is given to provide the reader with the background. An effective second order method, the Scaled Conjugate Gradient, is described and comments on the use of Genetic Algorithms for training are given. Finally three new error functions are derived: two *soft-monotonic* error functions based on the idea of making the classification more correct and one called the *MS-σ* error function which tries to eliminate the effect of local minima. In chapter 5 several other error functions are shown that are based on the idea of improving the generalisation ability of the network.

3.1 Introduction

Learning can be divided into two types: supervised$^{\heartsuit}$ and un-supervised$^{\heartsuit}$, depending on how much information is provided to the network. The learning paradigm considered here is **supervised learning**, which means that the network is provided with examples of both input pattern and output pattern. The desired output pattern is called the **target** and is denoted $\mathbf{T} = \{t_{L1}, t_{L2}, \ldots, t_{LN_L}\}$. Learning involves two important components: a measurement of the network's performance during learning and an algorithm that improves the performance. The measurement is normally done by an **error function**$^{\heartsuit}$ that gives a qualitative value of the error.

The most commonly used measurement is the Mean Square Error (MSE) function, given by:

$$E = \frac{1}{2P} \sum_{p=1}^{P} \sum_{i=1}^{M} (T_i^p - V_{Li}^p)^2 \tag{3.1}$$

where M is the number of output units and P is the total number of patterns.

The popularity of the MSE function is probably due to the fact that it is easy to understand as the squared distance and that it is mathematically manageable. Used in connection with estimation of the parameters of a classifier it will, given an asymptotically larger and statistically independent training set, produce an output that equals an optimal Bayesian discriminator [Duda and Hart 73] (pp. 154–155). The next paragraphs all assume the use of the MSE, because it is mathematically manageable and the methods will still be valid for other error functions.

There are several alternative error functions such as the relative entropy. A further description of this can be found in the references under the heading *"LEARNING AND ERROR FUNCTIONS"*.

3.2 Training of Feed-Forward Network

There are several techniques for training feed-forward networks. For a simple perceptron with output units having threshold transfer functions there is a special learning method named after the perceptron. It is called the perceptron learning rule$^{\heartsuit}$ [Rosenblatt 62]. This method is, however, restricted to being only valid if the problem is linearly separable. An extension of this method, known as the **pocket algorithm**$^{\heartsuit}$ [Gallant 86], is able to handle nonlinearly separable problems. The Pocket Algorithm has two nice properties: if the problem is linearly separable it will eventually be able to classify all patterns correctly; if the problem is not linearly separable it will in finite time find a solution where the output unit will produce a correct output on as many input patterns as possible. [1]

When the units in the network use transfer functions that are differentiable like sigmoid or Gaussian functions and an error function that is also differentiable it is possible to use a more mathematical approach. Training algorithms$^{\heartsuit}$ are often based on techniques for mathematical optimisation, such as the principle of Gradient Descent, and consist of repeatedly presenting the network with patterns from the entire set of patterns, usually hundreds of times. After each presentation of an input pattern, the network adjusts its weights so that if the same pattern is presented again, it will be more likely to produce the correct output pattern. One full presentation of all patterns in the training set is normally called an epoch. There are several algorithms with various advantages and drawbacks, which one to choose depends on the problem. There are also alternative approaches such as **Genetic Algorithms**, which can be used regardless of the transfer function and the error function. The focus in this chapter will, however, be on the more commonly used methods where the units' transfer functions are differentiable.

3.2.1 A Little Bit of Theory on Optimisation

In the part of the optimisation theory that will be considered in this thesis the goal is to find the variables that give the smallest value of a differentiable function. The basic idea is to get enough information about the function so that it will be possible to change

[1] The pocket algorithm is considered to be optimal in the sense that it finds the weights that give the maximum number of input-output relations contained in the training set. In [Muselli 95] it is shown that this only is true under certain conditions.

the variables from one point in the parameter space to another and eventually end up at a point where the function is at its minimum. The information about the function can be derived by calculating geometric properties of the function, commonly the gradient telling something about the slope of the function and/or the second derivatives (Hessian matrix) giving information about the curvature. These derivatives are normally used in connection with a Taylor approximation$^{\heartsuit}$ describing the error function. If the Taylor approximation uses only the gradient, the method is commonly called a 1st order method. If it also applies the Hessian matrix, it is called a 2nd order method.

Consider for example the function $f(x) = x^2 - 8x + 20$. At the start $x = 9$ and the goal is automatically to find an x where the value of $f(x)$ is smaller and eventually will be at the minimum. One strategy would be to calculate the gradient and then move x in small steps in the opposite direction and repeat this procedure until the minimum of $f(x)$ will eventually be reached. How long it takes before the minimum is reached will depend on how large the step is. The size of the step is normally called the *step-size* or *learning-rate*. A nice illustration of how to determine the learning-rate can be found in [Le Cun 93a]. An alternative would be to calculate where the second derivative is zero and move x immediately to that value. The example is illustrated in figure 3.1

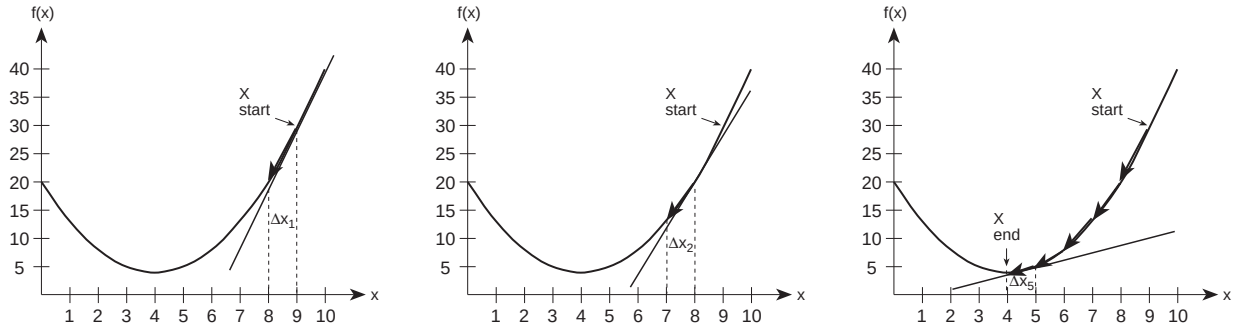


Figure 3.1: Evolution of x started from 9 while it finds the minimum of $f(x) = x^2 - 8x + 20$.

In neural network terms the function will be the error function and the variables will be the weights. However, the error function will in general not look like the illustration in figure 3.1 but more like the illustration in figure 3.2 where there are several minima (places having the property that if one moves away by a small distance, gradiet descent will make one come back). These minima are called **local minima** and the smallest of all the minima is called the **global minimum**. Many learning algorithms have problems with handling such functions and a general solution is not known. There are, however, some ways to remedy this problem, as shown later.

A further description of the *Optimisation Theory* can be found in e.g. [Fletcher 75] or [Gill *et al.* 81].

3.2.2 Gradient Descent

Using a 1st order Taylor approximation, with respect to the weights $\mathbf{w}$, the change in error with respect to the output unit's value can be expressed as:

$$E(\mathbf{w} + \triangle\mathbf{w}) \approx E(\mathbf{w}) + \triangle\mathbf{w}^T E'(\mathbf{w}). \tag{3.2}$$

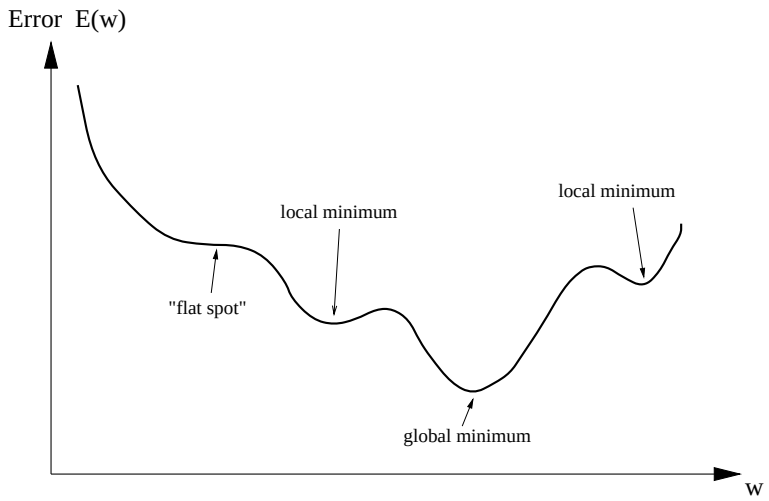


Figure 3.2: Illustration of a typical error function (also called **error landscape**).

A commonly used method for minimising this error function is the **Gradient Descent**. The Gradient Descent method guarantees that the error will decrease if the $\triangle \mathbf{w}$ is set equal $-\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ and η is small enough ($\eta > 0$). η is the *step-size* or *learning-rate* and as an alternative to choosing a fixed value for it, η can be determined by a line search method. If η is chosen optimally in each step, the method is often called the *steepest descent method*.

The method can be used in *off-line*$^{\heartsuit}$ or *on-line*$^{\heartsuit}$ mode. In the off-line mode the gradient vector is an accumulation of partial gradient vectors, one for each pattern in the training set. In the on-line mode, the gradient descent is performed successively on each partial error function associated with one given pattern in the training set. The update formula is then given by:

$$\triangle \mathbf{w} = -\eta \frac{\partial E^p(\mathbf{w})}{\partial \mathbf{w}} = -\eta E'^p(\mathbf{w}) \ , \quad \eta > 0, \tag{3.3}$$

where $\frac{\partial E^p(\mathbf{w})}{\partial \mathbf{w}}$ is the error gradient associated with pattern p. If η tends to zero over time, the movement in weight space during one *epoch* will be similar to the one obtained with one off-line update. In general, however, the learning rate has to be large to accelerate convergence, so that the paths in weight space of the two methods differ. The on-line method is often preferable to the off-line method when the training set is large and contains redundant information. This is especially true of problems where the targets only have to be approximated such as classification problems.

A nice theoretical discussion about issues concerning on-line and off-line techniques can be found in [Møller 93e](pp. 44–52). In practice there is sometimes a problem with the on-line methods when patterns from different classes are very differently distributed. If many patterns from a large class are presented to the network before patterns from other classes, the Gradient Descent method will probably find a weight position that is a minimum for this class. This weight position might not be the best position for the other classes and as the search for a new minimum continues it might turn out that it is not possible to change the weights so the error becomes smaller, and the method is

trapped in a local minimum. This problem might have been solved if another "path in the error landscape" had been followed. It is surprising that this problem has not been described very often in the neural network literature, probably because it is hard to handle theoretically or perhaps because the solution or more correctly a way to avoid this problem is simple. The trick to use is to ensure that data from the same class are not presented one after the other all the time and that the order of the data is changed from epoch to epoch. This technique is sometimes called **shuffle**$^{\heartsuit}$.

3.2.3 Simple Perceptron

If the feed-forward network is a simple perceptron with a differentiable transfer function $g(U_{Li}) = g(\sum_{j=0}^{N_0} w_{ij} V_{0j})$ the update of each weight is given by:

$$\triangle w_{ij} = -\eta \frac{\partial E(\mathbf{w})}{\partial w_{ij}} = -\eta (T_i^p - V_{Li}^p) g'(U_{Li}^p) V_{0j} \tag{3.4}$$

where η again is the step-size.

3.2.4 Radial Basis Function Network

The parameters in an RBF network can be determined in a number of ways. In earlier works like [Broomhead and Lowe 88] the parameters and patterns (input and output) were considered as a system of equations, normally overdetermined, and they were solved by methods like the Singular Value Decomposition method (SVD) [Golub and Loan 83]. The Gradient Descent methods can also be applied. Consider the case where output units $V_{Li} = g\left(\sum_{j=1}^{N_l} w_{ij}^L \Phi(U_{lj})\right)$ have differentiable transfer functions and the hidden units are Gaussian which delimits a hyper-ellipsoidal in the input space. The transfer function is then given by:

$$\Phi(U_{lj}) = e^{-\frac{1}{2}\sum_{k=1}^{N_0}\left(\frac{V_{0k}-c_{jk}}{\sigma_{jk}}\right)^2} \tag{3.5}$$

and the update of each parameter is given by:

$$\triangle w_{ij}^L = -\eta (T_i^p - V_{Li}^p) g'(U_{Li}^p) \Phi(U_{lj}) \tag{3.6}$$

$$\triangle c_{jk} = -\eta \sum_{i=1}^{N_L} \left((T_i^p - V_{Li}^p) g'(U_{Li}^p) w_{ij}\right) \Phi(U_{lj}) \frac{V_{0k} - c_{jk}}{\sigma_{jk}^2} \tag{3.7}$$

$$\triangle \sigma_{jk} = -\eta \sum_{i=1}^{N_L} \left((T_i^p - V_{Li}^p) g'(U_{Li}^p) w_{ij}\right) \Phi(U_{lj}) \frac{V_{0k} - c_{jk}}{\sigma_{jk}^3} \tag{3.8}$$

where w_{ij}^L is a weight between the $i'th$ output unit and the $j'th$ RBF unit, c_{jk} is the $k'th$ centre coordinate, and σ_{jk} is the $k'th$ radius.

3.2.5 Back-Propagation

Before 1978 the Gradient Descent was only applicable to single-layer feed-forward networks, because a method for calculation of the gradient $E^{'}(\mathbf{w})$ for multi-layer networks did not exist. A method which allows calculation of the derivative of the error between actual output and target with respect to all weights in the network was derived independently by several people, see for example [Rumelhart *et al.* 86] (chapter 8). The method is known as the *Error Propagation*$^{\heartsuit}$ or *Back-Propagation method*$^{\heartsuit}$, and is central to much current work on learning in neural networks. The fundamental idea is that the gradient $E^{'p}(\mathbf{w})$ of one particular pattern p, using the *chain rule*, can be calculated by one forward and one backward propagation.

The full Back-Propagation on-line procedure is as follows:

1. Initialise the weights to small random values.

2. Choose a pattern and propagate the signal forward through the network using

$$V_{li}^{p} = g(\sum_{j=0}^{N} w_{ij}^{l} V_{(l-1)j}^{p}) \tag{3.9}$$

   for each i and l until all the final output V_{iL} have been calculated.

3. Compute the deltas for the output layer

$$\delta_{Li} = g'(U_{li}^{p})(T_{i}^{p} - V_{Li}^{p}) \tag{3.10}$$

   by comparing the actual output V_{Li} with the desired target T_i for the pattern p being considered.

4. Compute the deltas for the preceding layers by propagating the error backwards

$$\delta_{(l-1)i} = g'(U_{li}^{p}) \sum_{j} w_{ij}^{l} \delta_{Lj}) \tag{3.11}$$

   for $l = L, L-1, ..., 2$ until a delta has been calculated for every unit.

5. Calculate

$$\triangle w_{ij}^{l} = \eta \delta_{(l-1)i} V_{li}^{p} \tag{3.12}$$

   and update all weights according to $w_{ij}^{new} = w_{ij}^{old} + \triangle w_{ij}^{l}$

6. Go back to step 2 and repeat for the next pattern.

The Back-Propagation algorithm can be applied to any feed-forward network regardless of the error function and transfer function as long as they are differentiable. In this thesis the transfer function is in most cases $g(U_{(l-1)j}^{p}) = Tanh(U_{(l-1)j}^{p})$, thus the derivative is:

$$g^{'}(U_{(l-1)j}^{p}) = g(U_{(l-1)j}^{p})(1 - g(U_{(l-1)j}^{p})) \tag{3.13}$$

3.2.6 Scaled Conjugate Gradient

Plain Back-Propagation suffers from a number of disadvantages, including slow convergence and the need to set one or more learning parameters, and many researchers have since used more sophisticated optimisation techniques to improve the learning behaviour. Several second order methods (using 2nd order Taylor approximation) have been developed (or adapted) for use with neural networks [Battiti 92]. Methods like the Quick-Prop[2] developed by [Fahlman 89] and the Conjugate Gradient are becoming increasingly popular.

The Scaled Conjugate Gradient algorithm developed by [Møller 93a] is a variation of a standard Conjugate Gradient$^{\heartsuit}$ algorithm. The major idea of Conjugate Gradient algorithms is that up to second order they produce non-interfering directions of search. This means that minimisation in one direction $\mathbf{d}_t$ followed by minimisation in another direction $\mathbf{d}_{t+1}$ imply that the error has been minimised over the whole subspace spanned by $\mathbf{d}_t$ and $\mathbf{d}_{t+1}$. The weight update formula is given by:

$$\triangle \mathbf{w}_t = \epsilon_t \mathbf{d}_t \tag{3.14}$$

where ϵ_t is the step-size and $\mathbf{d}_t$ is the search direction. The search directions are given by:

$$\mathbf{d}_{t+1} = -E^{'}(\mathbf{w}_{t+1}) + \beta_t \mathbf{d}_t \tag{3.15}$$

where $\mathbf{w}_t$ is a vector containing all weight values at time step t and β_t is:

$$\beta_t = \frac{|E^{'}(\mathbf{w}_{t+1})|^2 - E^{'}(\mathbf{w}_{t+1})^T E^{'}(\mathbf{w}_t)}{|E^{'}(\mathbf{w}_t)|^2} \tag{3.16}$$

In the standard Conjugate Gradient algorithms, the step-size ϵ_t is found by a line search$^{\heartsuit}$ which can be very time-consuming because this involves several calculations of the error and/or the first derivative. In the Scaled Conjugate Gradient algorithm, the step-size is estimated by a scaling mechanism thus avoiding the time-consuming line search. The step-size is given by

$$\epsilon_t = \frac{-\mathbf{d}_t^T E^{'}(\mathbf{w}_t)}{\mathbf{d}_t^T \mathbf{s}_t + \lambda_t |\mathbf{d}_t|^2} \tag{3.17}$$

where $\mathbf{s}_t$ is

$$\mathbf{s}_t = \frac{E^{'}(\mathbf{w}_t + \sigma_t \mathbf{d}_t) - E^{'}(\mathbf{w}_t)}{\sigma_t} \quad , 0 < \sigma_t \ll 1$$

ϵ_t is the step-size that minimises the second order approximation to the error function. $\mathbf{s}_t$ is a one-sided difference approximation of $E^{''}(\mathbf{w}_t)\mathbf{d}_t$. λ_t is a scaling parameter whose function is similar to the scaling parameter found in Levenberg-Marquardt methods. The parameter Δ_t defined by:

$$\Delta_t = \frac{E(\mathbf{w}_t) - E(\mathbf{w}_t + \epsilon_t \mathbf{d}_t)}{E(\mathbf{w}_t) - E_q(\mathbf{w}_t + \epsilon_t \mathbf{d}_t)} \tag{3.18}$$

[2] A semi-second order method using the gradient at the current and previous weight positions to estimate the Hessian.

where $E(\mathbf{w}_t)$ is the real error and $E_q(\mathbf{w}_t)$ is the quadratic approximation of the error, controls the updating of λ_t. λ_t is in each iteration raised or lowered according to how close Δ_t is to 1, i.e. how good the second order approximation is to the real error.

SCG exhibits second order convergence properties and contains no problem dependent parameters. See [Møller 93a] for a detailed description of SCG.

3.2.7 Training by Genetic Algorithms

Genetic Algorithms (GAs)$^{\heartsuit}$ are general search techniques, based on biological principles ([Goldberg 89] [Holland 75]). In many cases GAs have shown to be effective adaptive techniques for optimisation. GAs have been applied in conjunction with neural nets to provide a learning algorithm and to find the optimum architecture of the network in terms of number of hidden units and their connectivity. The considerations here are limited to the first case, while the latter will be discussed briefly in chapter 6. GAs are suitable for finding "good areas" in the weight space, because GAs are global methods. A review of some of the work done by using GAs to adjust the weights can be found in e.g. [Schiffmann *et al.*92]. They also conclude that GAs are only able to work with less complex networks (< 50 units). Recent work at the Computer Science Department at Aarhus University by [Korning 94] has confirmed this conclusion, but [Schmidt and Stidsen 94] have also shown that by constraining the value of the weights and using weight sharing it is possible to get good performances.

3.3 Soft-Monotonic Error Function

Error functions like mean square are known to be Bayes optimal$^{\heartsuit}$ in the sense that minimisation with these functions produces solutions that approach the greatest lower bound on generalisation error as the training set approaches infinity. But when the training set is small, this approximation can be poor [Buntine and Weigend 91b], and it is necessary to impose constraints on the network solutions. This is in a Bayesian perspective the same as choosing appropriate priors which is also strongly related to penalty terms or regularisers in statistical literature, see chapter 5.

In order to illustrate a problem with the mean square error function, consider a network with two output units having outputs between 0 and 1. The outputs are mapped onto an V_{L1}-axis and a V_{L2}-axis respectively as shown by the simple illustration [Hampshire 92] in figure 3.3.

If the desired target pattern is (1 0) then all outputs to the right of the line $V_{L2} = V_{L1}$ in figure 3.3. can be considered to be correct. If the mean square error function is used the error in point x_1 belonging to the opposite class will be lower than the error in x_2 which belongs to the right class. Only if the contours of equal errors are straight lines parallel to $V_{L2} = V_{L1}$, no regions exist with misclassification and lower error than other regions with correct classification. Hampshire defines error functions that satisfy such a condition to be *monotonic*. Hampshire strongly suggests that non-monotonic behaviour in training can cause the often seen "over-learning", i.e. where the recognition performance

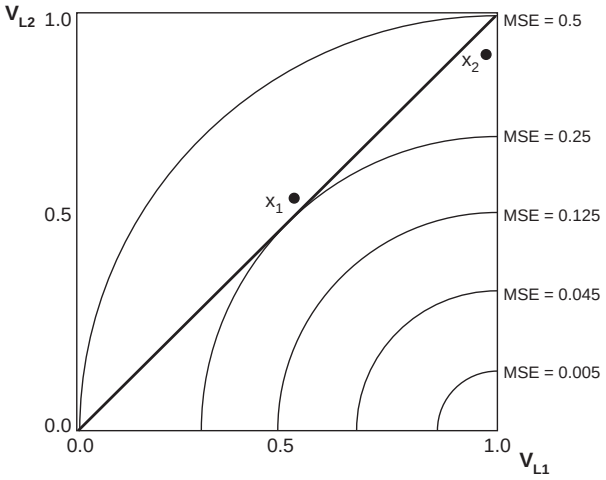


Figure 3.3: Illustration of *non-monotonicity*. The V_{L1}-axis and V_{L2}-axis correspond to the output from a network with two output units. The curves correspond to regions with equal mean square error on target pattern (1 0).

on a disjoint test set peaks and then degrades, while training set performance continues to improve. This is described further in chapter 4.

The problem with suboptimal solutions exists in the form of local minima or most often flat regions in error space. Suboptimal solutions in flat regions are often characterised by having a few patterns wrongly classified and many correctly classified. The regions are flat because the network gradients are small for extremely wrong outputs. Minimisation of the mean square error function might very well converge to such regions because the training algorithms are *greedy* algorithms, updating weights in the direction of fastest error decrease, and no mechanism in the error function prevents the update of weights into these regions.

In the following paragraph two new error functions are defined that satisfy a *soft-monotonic* condition in the sense that the functions are asymptotically monotonic within the limit of certain parameters associated with the functions.

3.3.1 Imposing Constraints on Network Solutions

Instead of insisting on strict monotonicity, it is possible to define error functions that satisfy a *soft-monotonic* condition, where a certain parameter controls the *degree* of monotonicity. The main idea is to incorporate appropriate constraints into the error function, so that the weights are constrained away from bad regions in weight space.

A way to avoid suboptimal solutions is to strictly minimise the number of misclassifications. Hampshire defines such an approach that works for classification problems [Hampshire 92]. A more general approach that involves a soft minimisation of misclassifications is presented here.

Since good solutions are characterised not only by low average error but also by having as many patterns with low error as possible, a good idea would be to include both terms

in the error function. One possible approach is to define an error function that penalises errors of large magnitude. The first approach is called the Exponential Error function (see e.g. [Møller *et al.* 94] or [Møller 93e] where it is first described), defined as:

$$E(\bar{w}) = \frac{1}{2}\sum_{p,j} e^{-\alpha(V_j^p - t_j^p + \beta)(t_j^p + \beta - V_j^p)} \tag{3.19}$$

where α and β are positive parameters. The derivative to (3.19) with respect to a given V_j^p is

$$\frac{dE(\bar{w})}{dV_j^p} = -\alpha(t_j^p - V_j^p)e^{-\alpha(V_j^p - t_j^p + \beta)(t_j^p + \beta - V_j^p)} \tag{3.20}$$

It is easy to see that the global minimum for (3.19) is when $t_j^p = V_j^p$, $\forall p, j$. The function of α and β is illustrated in figure 3.4.

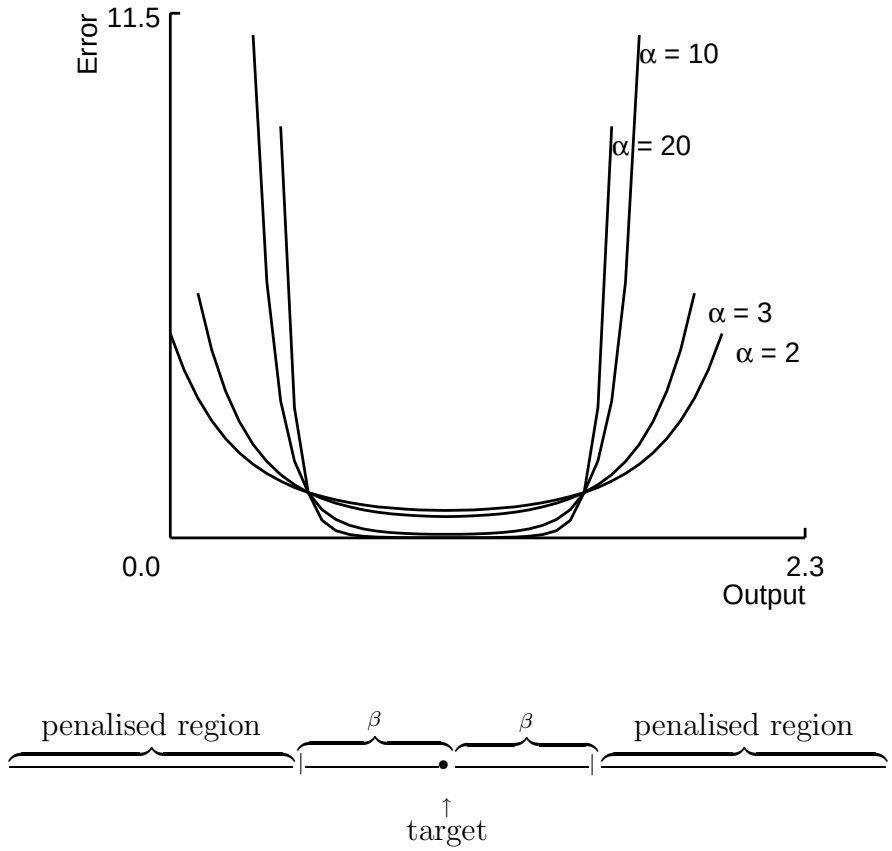


Figure 3.4: The function of the α and β parameter.

β defines the width of the acceptable error around the desired target and α controls the steepness of the exponentially growing error in the penalised regions outside the interval. If α is small, equation (3.20) resembles the derivative of the mean square function. But the higher α gets, the more the constraint imposed on the penalised regions is active. When there are no errors in the penalised regions, β is decreased, so that the outputs are pulled towards the targets. A high α value gives large partial error derivatives inside the penalised

regions and small partial error derivatives when outside the regions. So the higher the α value gets, the more the errors will tend to arrange themselves inside the region around the target. This gives a sort of balanced distribution of errors. For regression problems it is well known in statistics that a balanced set of errors can yield better generalisation, this is often referred to as *variance heterogeneity* [Seber and Wild 89]. It is an open question whether this is also true in connection with classification problems.

When α increases to infinity, the exponential error function is monotonic. Surely, for a fixed number of patterns in the training set, a large enough α can be selected so that the error function is monotonic. The problem is how large α should be to ensure monotonicity in a given problem. Selecting too high an α slows down the convergence, because of too hard constraints imposed on the acceptable paths down to the minimum. On the other hand, too small an α results in non-monotonic behaviour of the error function. One promising approach would be to adapt α similarly to the penalty parameters in constrained optimisation, starting with a small α and then successively increasing α during training. This approach has not been tried yet. It seems that just setting α to a "reasonable" size yields good results.

Notice that the exponential error function also works for non-classification problems and that the soft-monotonicity condition can be obtained for any accuracy required by adjustment of the β parameter.

A more direct way of balancing errors is to minimise the variance of the magnitude of the errors. This can be done by adding the variance as a penalty term to an existing error function like mean square. This approach is called Variance Error function.

$$E(\bar{w}) = \frac{1}{NP}\sum_{j}^{N}\sum_{p}^{P}(t_j^p - V_j^p)^2 + \eta\frac{1}{NP}\sum_{j}^{N}\sum_{p}^{P}\left((t_j^p - V_j^p)^2 - \tfrac{1}{PN}\textstyle\sum_i^N\sum_q^P(t_i^q - V_i^q)^2\right)^2 \quad (3.21)$$

where η is a positive penalty parameter, N the number of output units and P the number of patterns. The derivative to (3) is:

$$\frac{dE(\bar{w})}{dV_j^p} = -\frac{1}{NP}(t_j^p - V_j^p)\Big(2 + 4\eta\frac{PN-1}{PN}\big((t_j^p - V_j^p)^2 - \tfrac{1}{PN}\textstyle\sum_i^N\sum_q^P(t_i^q - V_i^q)^2\big)\Big) \quad (3.22)$$

From (3.20) it can be observed that while the exponential error function can be used in both on-line and off-line training mode, the minimum variance error function can only be applied in off-line mode.

A number of experiments with these two error functions (see *Appendix D*) have shown that minimisation with the new error functions produces on average better solutions with respect to generalisation than the mean square error function.

3.4 MS-σ Error Function

In the previous paragraph it was mentioned that suboptimal solutions exist in the form of very flat regions in error space and that the reason for this is that the network gradients are small for extremely wrong outputs. Another situation causing local minima, also called *flat spots*, occurs when the output units of the network are trained to saturation$^{\heartsuit}$. Recall the update equation in (3.4) given by:

$$-\frac{\partial E(U_{Li}^p)}{\partial w_{ij}} = -\frac{\partial E(U_{Li}^p)}{\partial U_{Li}^p}\frac{\partial U_{Li}^p}{\partial w_{ij}} = (T^p - V_{Li}^p)(g'(U_{Li}^p))V_{(L-1)j} \tag{3.23}$$

where E is the MSE, T is the target (desired output), U_{Li} and V_{Li} are the activity and output respectively from the i'th unit in the output layer L, and $g()$ is the transfer function of the output unit.

If the units in the network are sigmoid-like and the output units are trained to saturation, the derivative of the transfer function g' will be zero, regardless of the value of the other terms of equation (1). This means that a flat spot area is reached.

3.4.1 Preventing Saturation (or Eliminating "Flat Spots")

In order to prevent saturation and speed up the learning procedure, Fahlman [Fahlman 89] introduced a small offset, $\sigma > 0$, that is added to the derivative. In the Gradient Descent method this corresponds to updating the weights proportional to:

$$-\frac{\partial E(U_L^p)}{\partial w_i} = -\frac{\partial E(U_{Li}^p)}{\partial U_{Li}^p}\frac{\partial U_{Li}^p}{\partial w_{ij}} = (T^p - V_L^p)(g'(U_L^p) + \sigma)V_{(L-1)j} \tag{3.24}$$

This simple idea is very effective, but it can only be directly applied to first order learning methods. At first this seems very harmless, but a closer look at this equation reveals that the "error surface" has been changed. There is no direct correspondence between the derivative and the actual MSE anymore. Therefore the method cannot be used in connection with second order methods, unless the "new" error function is found. The reason is that second order methods often require knowledge of the error itself or of the second derivative (Hessian matrix) calculated from the error. However, the error function that corresponds to the derivative in (3.24) can be derived by simple integration. Solving the $\frac{\partial E(U_L^p)}{\partial U_L^p}$ part of the differential equation in (3.24) the error becomes:

$$E(U^p) = \frac{1}{2}(T^p - V_L^p)^2 + \sigma\Big(G(U_L^p) - T^p U_L^p\Big) \quad \text{where } G(U_L^p) = \int g(U_L^p)\partial U_L^p$$

The first term is equal to the "old" error. If $g(U) = Tanh(U) \Rightarrow G(U) = ln(Cosh(U)) + C$. In figure 3.5. the error function with offset, which from now on is referred to as MS-σ, is shown for different offset values.

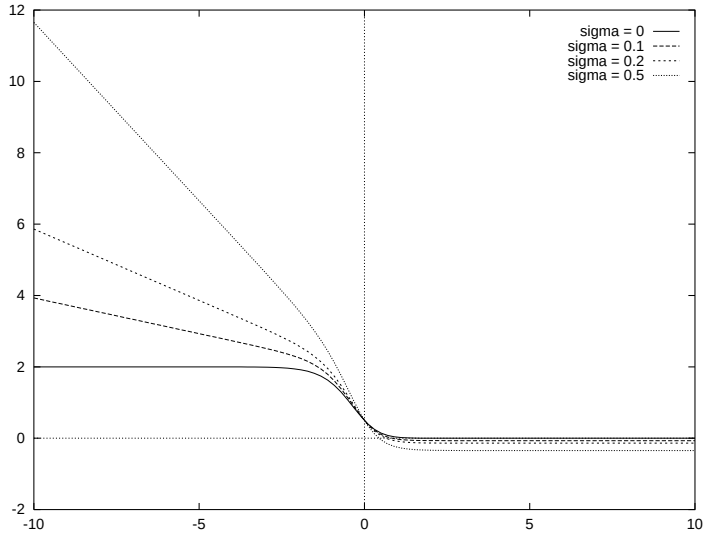


Figure 3.5: Error function graphs for one pattern with target value 1. The x-axis corresponds to the activity U^p and the y-axis to the error. The standard MSE function corresponds to $\sigma = 0$. Notice that the flat area in the direction of minus the target is eliminated when $\sigma > 0$.

For $\sigma = 0$, the error has two flat areas, one when $g(U_L^p)$ has the same sign as the target, and another when $g(U_L^p)$ has the opposite sign. It is obviously desirable to eliminate the latter flat area, which is exactly what is achieved when $\sigma > 0$. When the error in the new error surface is in a minimum, $\frac{\partial E^p}{\partial U_L^p} = 0$, and the difference between the target T^p and the output V_L^p is not zero, $g'(U_L^p)$ is equal to the offset term. The effect of this is that the unit is not trained to saturation, which is often a difficult situation to escape. The same result could have been achieved if other error functions more suitable than MSE were used, e.g. Soft-Monotonic Error Function.

The idea of adding a term to the gradient is as illustrated very effective, and calculating the new error function, and using this makes it applicable to higher order learning algorithms, including the SCG.

In one particular case it is very important to avoid saturation: when a network is trained and built at the same time by the Cascade-Correlation Algorithm (CCA) [Fahlman and Lebiere 90]. A detailed description of the CCA can be found in chapter 6. Experiments with the CCA in *Appendix E* using the SCG as the learning algorithm showed that SCG combined with MS-σ error function performs much better than when combined with MSE without an offset. Since second order methods like SCG are known to yield faster convergence than standard Back-Propagation, it is extremely important that this combination has been made possible.

3.5 Résumé and Concluding Remarks

This chapter has described the learning algorithms and error functions that will be used and have been applied in this thesis. The learning algorithms based on the Gradient Descent are shown for a simple perceptron, RBF network and MLP, including the Back-Propagation and the Scaled Conjugate Gradient. Three new error functions were introduced: the Exponential and the Variance error functions, based on the idea that Soft-Monotonic Error Functions produce a more correct classification, and the MS-σ error function that makes it possible to apply the offset approach with second order training algorithms like the Scaled Conjugate Gradient algorithm. The offset approach is able to speed up the learning but what is more important it prevents the network's output units from being trained to saturation.

Chapter 4

Generalisation

This chapter deals with the most essential issue for neural networks as well as for any learning machine: generalisation. In order to understand the meaning of generalisation, how to estimate it and to point out some problems with measuring generalisation, various definitions including a graphical illustration of a curve-fitting problem are provided. The focus is on the *VC-theory* and the framework of *Probably Approximately Correct (PAC) learning*. Both the *VC-theory* and the *PAC* theory are based on the measurement of a machine's capacity in terms of *growth function* and *VC-dimension* ($\mathcal{VC}_{dim}$), which are in general unknown. An exception is the linear classifier and it is shown that many of the self-constructing algorithms that have been developed during the last few years have the same $\mathcal{VC}_{dim}$ as a linear classifier.

4.1 Introduction

The question of generalisation is really the most important issue for learning machines$^{\heartsuit}$ as it deals with the ability to find the underlying rules. The mathematical underpinnings on these issues are rather weak, but various strong attempts are being made to attack these questions. Before more strict mathematical definitions of generalisation - or more precisely the generalisation error - are given, it might be a good idea to get an introduction to this field.

Consider first the example where the task for a learning machine is to calculate the sum of two numbers. When several training patterns have been shown and the parameters of the machine have been adjusted successfully, the training set will have been learned. The situation might be that the machine has discovered that the underlying rule is simply the arithmetic rule of adding, but the case might also be that the machine has just made a table of the training examples, and will fail badly when a pattern outside the training set is presented. In the latter case the memory capacity of the machine has to be large.

Another way of showing this problem is the very illustrative example of curve-fitting. Figure 4.1 shows an example where the training set in c) is given by the underlying truth in a) disturbed by the noise in b).[1]

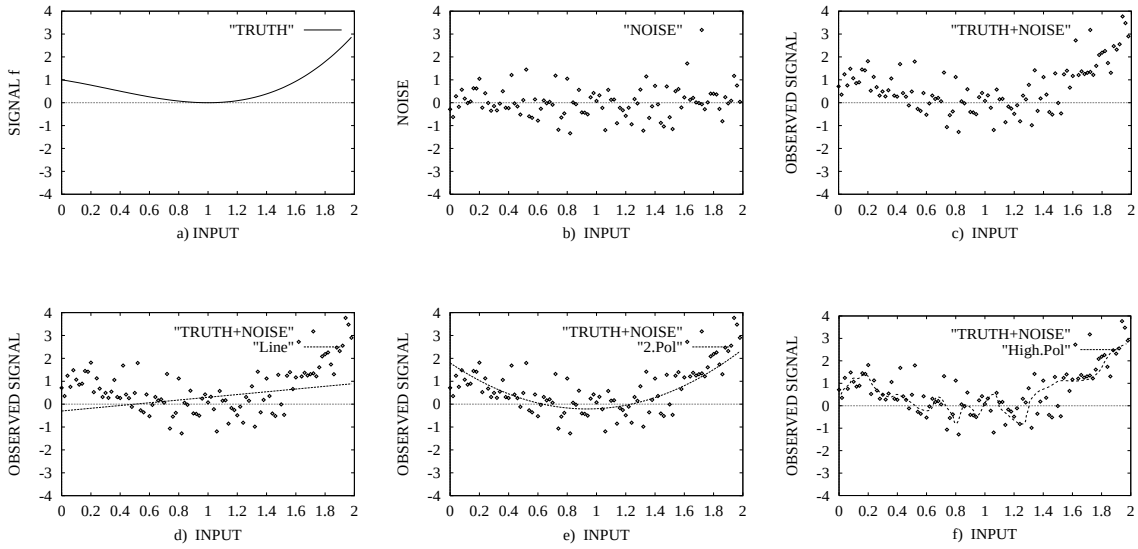


Figure 4.1: Illustration of a curve-fitting problem. a) The underlying truth. b) The noise. c) Training set (a) + b)). d) A model with too few parameters (under-fitting). e) A model with a suitable number of parameters (good-fit), and f) A model with too many parameters (over-fitting).

The figures d), e) and f) show the results from using a model with too few parameters (under-fitting), a model with a suitable number of parameters (good-fit), and finally a model with too many parameters (over-fitting).

Consider figure 4.1 again and imagine that the mean square error between the training set and the models were calculated: there is no doubt that the model with too many parameters would have the smallest error and therefore it must be the best. This is, however, not the case because the best model is the one whose error is smallest in proportion to the underlying truth. If the mean square error was calculated between the underlying truth and the three models, the model of the polynomial of 2nd order in b) would be the best. There are several methods that try to control this trade-off between the number of parameters in the model and the mean square error between the training set and the model's output. Some of these methods are shown later in this chapter and in chapter 5.

The general expressions like learning machine and model have been used to stress the fact that generalisation is a problem that is not only of interest for neural networks. The two examples deal with function approximation or more generally a mapping from input to output, which is exactly what a feed-forward neural network can perform. The parameters in the approximating polynomial correspond to the weights in a network while the **complexity** and **capacity** are somehow related to the number and topology of units

[1]This way of considering a set of given data, as the sum of the true function and unknown function f^d and some noise n, is one of the most commonly used models.

and weights, and thereby the network's architecture. It is therefore not surprising that the network's architecture has an influence on the generalisation ability.

There are many references with a similar approach to explain generalisation, e.g. [Geman *et al.* 92] providing an excellent link between the intuitive approach and a more mathematical description from a statistical viewpoint.

4.2 Capacity and Complexity

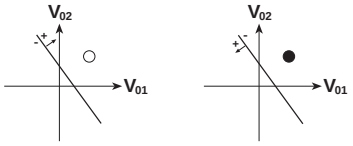
In the previous paragraph the connection between the performance of a neural network and its complexity and capacity was indicated. The following description will start with a general explanation of the terms and then go on by providing some basic definitions and a measurement of the capacity known as the *VC-dimension* ($\mathcal{VC}_{dim}$). By means of an example of the linear classifier[2] applied to binary classification problems, a graphical illustration of the concepts *growth function* denoted $\triangle(\cdot)$ and $\mathcal{VC}_{dim}$ will be given.

The *complexity*$^{\heartsuit}$ of a feed-forward neural network is related to the number and topology of units and weights. In general one would say that a multi-layer perceptron is more complex (has a higher degree of complexity) than a simple perceptron. The *capacity*$^{\heartsuit}$ expresses how many different functions a network with a given architecture can implement. It might seem that higher complexity means higher capacity and the two terms are therefore often confused. There is, however, a crucial difference. There are several networks where some of the weights are not allowed to change, e.g. networks built by construction algorithms. The complexity of these networks is still the same, regardless of whether the weights can change or not, but the capacity is reduced. It is common then to talk about the capacity as the number of **free parameters** i.e. the number of weights that can be adjusted. For a linear classifier this connection is true, but in general as soon as nonlinearity is introduced it is not valid. This will be shown in a little while.

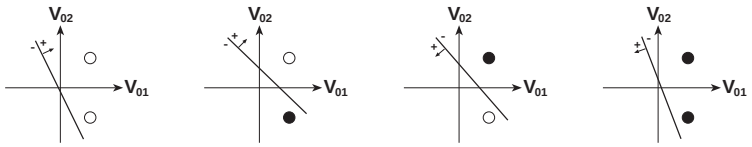
4.2.1 Growth Function $\triangle(\cdot)$ and VC-Dimension $\mathcal{VC}_{dim}$

Given a network it will be possible by varying the parameters w to find a number of different binary functions$^{\heartsuit}$ so that a set of P unlabelled patterns $\{\mathbf{V}_0^p, 1 \leq p \leq P\}$ will be classified correctly. The **growth function**$^{\heartsuit}$ $\triangle(P)$ is defined as the maximum number of different binary functions that can be implemented by the network on a data set containing P patterns. If P is small $\triangle(P) = 2^P$, because the number of binary functions for P elements is 2^P. Vapnik and Chervonenkis (1971) have shown that this exponential growth will continue until a certain characteristic value, where the growth will become polynomial. The characteristic value is the **VC-dimension**$^{\heartsuit}$ denoted $\mathcal{VC}_{dim}$. So $\mathcal{VC}_{dim}$ describes the maximum number of unlabelled patterns for which all possible binary labelling can be learned without error. To make this more clear consider the examples of a linear classifier, $f_W(\mathbf{U}_L) = sgn(\sum_{i=1}^{N} w_{1i}\mathbf{V}_{0i} + w_{10}V_{00})$ illustrated in figure 4.2.

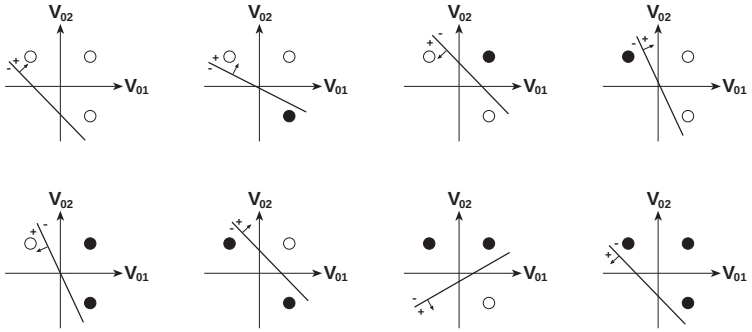
[2]A linear classifier is the generic name for a classifier that uses a linear combination of the input to decide which class a given input belongs to. When implemented as a neural network with no hidden layers it is called a simple perceptron or linear threshold network.



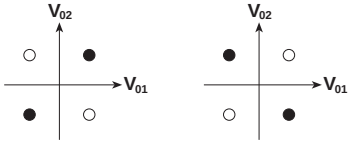
A single pattern - 2 possibilities, $\triangle(1) = 2$



2 patterns - 4 possibilities, $\triangle(2) = 4$



3 patterns - 8 possibilities, $\triangle(3) = 8$



4 patterns - 16 possibilities [3]. These two patterns
could not be learnt, therefore $\triangle(4) = 14$ and $\mathcal{VC}_{dim} = 3$

Figure 4.2: Illustration of the functional capabilities for a two input single output linear classifier with different numbers of patterns. White indicates that the pattern belongs to class 1 and black that it belongs to class -1.

Consider figure 4.2. By plotting the number of functions that the classifier is able to learn correctly against the number of training patterns, the following graph of the growth function can be generated, see figure 4.3.

The figure shows that the growth function $\triangle(P) = 2^P$ will be valid until the number of patterns exceeds the $\mathcal{VC}_{dim}$ whereupon it becomes polynomial. In [Vapnik 82] it is shown that a bounding function for the growth function when $P \geq \mathcal{VC}_{dim}$ is:

$$\triangle(P) \leq 1.5 \frac{P^{\mathcal{VC}_{dim}}}{\mathcal{VC}_{dim}!} \tag{4.1}$$

The graph of the growth function is not only valid for the two input classifiers, but

[3] This example is the well-known XOR problem, which cannot be solved because a linear classifier is only able to correctly classify patterns that are linearly separable.

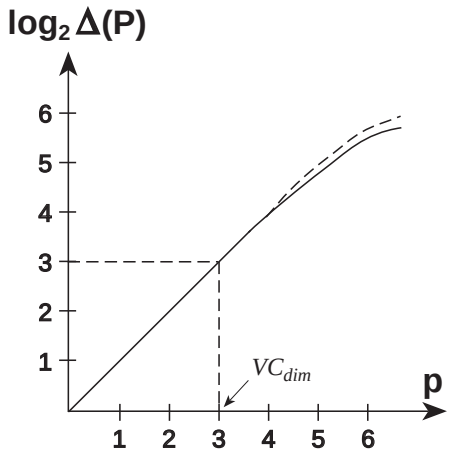


Figure 4.3: The growth function. The dotted line shows a bounding function for the growth function when $P \geq \mathcal{VC}_{dim}$, $\triangle(P) \leq 1.5P^{\mathcal{VC}_{dim}}/\mathcal{VC}_{dim}!$.

counts for all linear classifiers. Mathematical proofs of the bounding function in the case of general linear classifiers can be found in [Hertz *et al.* 91], [Cover 65] and [Nilsson 89].

From this description one could get the idea that the capacity is simply equal to the number of adjustable weights and as such proportionally related to the complexity. In order to illustrate that this is not in general the case consider a single input network with the following mapping function $f_W(U_L) = sgn(sin(w_1V_{01} + w_0V_{00}))$. In this case it will be possible to implement all 2^P binary functions by varying w_1, which plays the role of a frequency. Since $\triangle(P) = 2^P$ for all P, regardless of the distribution from which the input patterns are drawn, the capacity is infinite, although the function only contains two parameters. Note that the infinite capacity means that all patterns could be learned, just like the "adding" example in the introduction, so the network has just memorised the patterns. As it will be shown later the generalisation ability will in cases of infinite capacity be poor.

4.3 Definition of Generalisation (Error)

The most common way to define a neural network's generalisation ability$^{\heartsuit}$ is often to define its opposite equivalent, the generalisation error$^{\heartsuit}$.

The error between the output $\mathbf{V}_L = f_W(\mathbf{V}_0)$ and the desired output (target) $\mathbf{T} = f^d(\mathbf{V}_0)$, where f^d is the function representing the true but unknown function, is normally calculated by an error function $E(\mathbf{V}_L, \mathbf{T})$. The learning error on a set of known data, the training set $\{(\mathbf{V}_0^p, \mathbf{T}^p), 1 \leq p \leq P\}$, is defined as:

$$E_L(\mathbf{W}) = \frac{1}{P}\sum_{p=1}^{P} E(\mathbf{V}_L^p, \mathbf{T}^p) \tag{4.2}$$

and measures the dissimilarity between f_W and f^d, within the restricted domain of input patterns in the training set. Note that it is a function of weights in the weight space $\mathbf{W}$.

The $E(\mathbf{V}_L, \mathbf{T})$ used in $E_L(\mathbf{W})$ is often, but not necessarily, the Mean Square error as shown in chapter 2.

The expected error on unknown data is called the generalisation error. It is defined as an average over the full distribution of input-output pairs and can be expressed as:

$$E_G(\mathbf{W}) = \int E(\mathbf{V}_L, \mathbf{T}) P(\mathbf{V}_0, \mathbf{T})\, d\mathbf{V}_0\, d\mathbf{T}. \tag{4.3}$$

where $P(\mathbf{V}_0, \mathbf{T})$ is the joint probability distribution formed by $P(\mathbf{V}_0)$ describing the distribution of data in the input space and $P(\mathbf{T}|\mathbf{V}_0)$ describing the functional dependence, so $P(\mathbf{V}_0, \mathbf{T}) = P(\mathbf{T}|\mathbf{V}_0)P(\mathbf{V}_0)$. The generalisation error is the expectation value of error for an arbitrary $(\mathbf{V}_0, \mathbf{T})$ point drawn from the $P(\mathbf{V}_0, \mathbf{T})$ distribution.

The real goal of learning is to minimise the generalisation error. But the joint probability distribution $P(\mathbf{V}_0, \mathbf{T})$ is unknown and the only available information is contained in the training set.

4.4 Review of Generalisation Approaches

In order to solve the problem of not knowing the joint probability distribution $P(\mathbf{V}_0, \mathbf{T})$, various methods based on distinct underlying frameworks have been developed. Basically, however, they are all built on estimating the generalisation error by an error computed empirically on the basis of data available in the form of a data set.

The most common approach used to get information about data that the network has not seen is known as **Cross Validation**$^\heartsuit$. The basic idea is to split the data set into two, one part of data for training and another only used for test, so $P = P_{Training} + P_{Test}$. The generalisation error is simply approximated as:

$$E_G(\mathbf{W}) \approx \frac{1}{P} \sum_{p=1}^{P_{Test}} E(\mathbf{V}_L^p, \mathbf{T}^p) \tag{4.4}$$

The $E(\mathbf{V}_L, \mathbf{T})$ used in $E_G(\mathbf{W})$ is for classification problems often some sort of **Indicator function**$^\heartsuit$ which means that $E(\mathbf{V}_L, \mathbf{T}) = 0$ if $f_W(\mathbf{V}_0) = \mathbf{T}$, and $E(\mathbf{V}_L, \mathbf{T}) = 1$ otherwise. So $E_G(\mathbf{W})$ is the frequency of error on the test set.

A crude illustration of the connection between the error on the training set and the test set as a function of epochs (number of times the full training set is presented to the network) is shown in figure 4.4.

The crucial advantages of Cross Validation are that it is easy to use, the user does not need to know anything about the distribution of the data, and what is most important it allows the use of two error functions. During training it is often desirable to use a differentiable error function so methods like Back-Propagation can be applied to minimise some quantity measurement for the error. However, the interesting thing in connection with classification problems is the number of correct classifications, so in the test phase some form of indicator function can be applied.

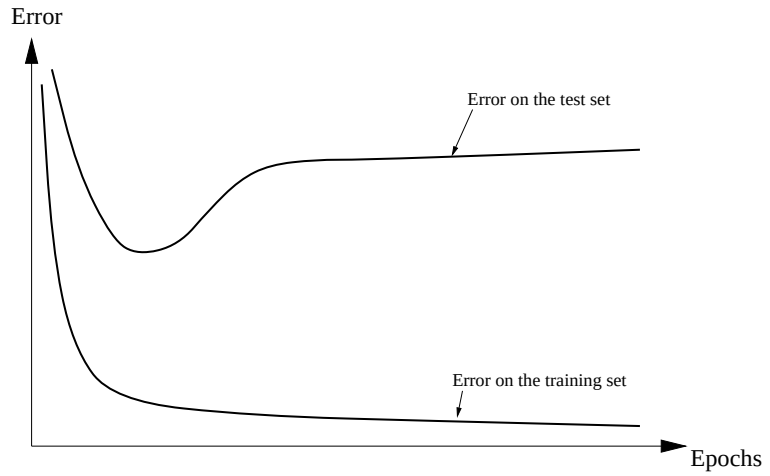


Figure 4.4: Illustration of the error on the training set and the test set as a function of epochs

One of the open questions with this method is how to split the data set in such a way that the result will be reliable. Of course one would like the training set to be large in order to learn as much as possible, but then the test set cannot be expected to represent all unknown data very well. A special version of the Cross Validation method called *leave-one-out*$^{\heartsuit}$ can help eliminate this problem. The leave-one-out method uses all training data except one for training and then tests the performance of the single data. This process is continued until each training pattern has been used as test pattern. The average of all test errors is then used as an estimate for the generalisation error. An excellent description of the method and mathematical proof can be found in [Fukunaga 90] (page 220). A big drawback of this method is that the network has to undergo a full training phase for all training sets. A way to soften this problem is simply to take larger blocks of patterns for testing. A special version of the Cross Validation method where the training set is further split into a new training set and a validation set is described in chapter 5.

Another useful approach comes from statistical mathematics, where the idea is that the generalisation error is a sum of two components: the error on the training set and an additional term depending on the network's complexity. This is known as regularisation$^{\heartsuit}$ and approximates $E_G(\mathbf{W})$ in the following way:

$$E_G(\mathbf{W}) \approx \sum_{p=1}^{P} E(\mathbf{V}_L^p, \mathbf{T}^p) + \lambda C(f_{\mathbf{w}}) \tag{4.5}$$

where $C(f_{\mathbf{w}})$ is a monotonic increasing cost-function of the mapping function $f_{\mathbf{w}}$ as the complexity of $f_{\mathbf{w}}$ increases. λ is the regularisation parameter that controls the compromise between the degree of smoothness of solution and its closeness to the data. One of the advantages of this approach is that all available data can be used for training.

This approach has been used in connection with RBF networks ([Poggio and Girosi 90a] [Poggio and Girosi 90b]) which will be shown in chapter 5 in connection with optimisation of the architecture.

References to descriptions of alternative approaches can be found in the bibliography under the heading *GENERALISATION* including a Bayesian approach by [MacKay 92]

[MacKay 91], Akaike's Final Prediction Error by [Fogel 91], [Rasmussen 93], Expected Test Set Error and Risk Minimisation by [Moody 92],[Moody 94], [Fogel 91] and Minimum Description Length Approach by [Rissanen 84].

4.5 Theoretical Analysis of Generalisation

In the field of theoretical analysis of generalisation there are only two approaches that have made a significant impact to date: the approach based on statistical physics [4] and the approach based on computational learning theory. The first approach normally deals with *average case*, while the latter deals with *worst case*. Which approach to be preferred depends of course on application and personal attitude. The rest of this chapter will focus on the computational learning theory which includes the *VC-theory* and the *Probably Approximately Correct (PAC) learning theory*. Although the two approaches originate from different backgrounds, they are similar. The objective of both approaches - seen from a neural network point of view - is that they try to solve the questions:

> How many training patterns P are needed in order for the network to learn the true function f^d?
>
> Does a network exist that is able to learn f^d from examples within reasonable time, that is with polynomial computational complexity?

The answers to these questions assume that optimal training can be achieved, in the sense that there are no problems with local minima.

4.5.1 The VC Theory (Uniform Convergence)

The VC theory or perhaps it should be called the *Theory of Uniform Convergence of Empirical Risk*$^\heartsuit$ *to Actual Risk* was developed in the 70's and 80's by Vapnik. It includes a description of necessary and sufficient conditions as well as bounds to the rate of convergence [Vapnik 82] (chapter 6). These bounds, which are independent of the distribution function $P(\mathbf{V}_0, \mathbf{T})$, are based on a quantitative measure of the capacity of the set of functions implemented by the network: the *VC-dimension* of the set. The following description deals with the idea of uniform convergence, the concept of the VC-dimension is described paragraph 4.2.1.

The problem considered here is the classification problem where data from an N-dimensional input space are to be classified as belonging to one of M classes, which means that $\mathbf{V}_L \in \{-1, 1\}^M$. The error function $E(\mathbf{V}_L, \mathbf{T})$ is assumed to be an indicator function so that $E(\mathbf{V}_L, \mathbf{T}) = 0$ if $f_W(\mathbf{V}_0) = \mathbf{T}$, and $E(\mathbf{V}_L, \mathbf{T}) = 1$ otherwise. So $E_G(\mathbf{W})$ is the probability of error, and $E_L(\mathbf{W})$ is the frequency of error on the training set.

Inspired by the Bernoulli theorem, Vapnik and Chervonenkis (in their most famous paper [Vapnik and Chervonenskis 71]) and Vapnik ([Vapnik 82], [Vapnik 92]) defined the uniform convergence in the following way:

[4] For those who would like to know more about the statistical physics approach, [Hertz *et al.* 91], [Tishby 95] including references, provide a good introduction.

$$P(\sup_{\mathbf{w}\in\mathbf{W}} |E_G(\mathbf{W}) - E_L(\mathbf{W})| > \epsilon) \longrightarrow 0 \quad \text{as} \quad P \longrightarrow \infty \tag{4.6}$$

where "sup" denotes supremum, $\mathbf{W}$ is the space of all possible weight combinations in a network (also called the weight space), and ϵ is the error.

One very important thing about the *VC theory* is that it provides a bound to the rate of uniform convergence through the definition of the *VC-dimension* ($\mathcal{VC}_{dim}$). This term was described in paragraph 4.2.1. For a given network able to perform a mapping from $R^N \rightarrow B^M$ so that E_G can be expected to be small, the following inequality with probability $(1-\eta)$, simultaneously for all $\mathbf{w} \in \mathbf{W}$ will be valid[5] :

$$E_G(\mathbf{W}) \leq E_L(\mathbf{W})) + C\left(\frac{P}{\mathcal{VC}_{dim}}, E_L(\mathbf{W}), \eta\right) \tag{4.7}$$

where $C(\frac{P}{\mathcal{VC}_{dim}}, E_L(\mathbf{W}), \eta)$ is a confidence interval, a function which depends on P, the number of training patterns, $\mathcal{VC}_{dim}$ the capacity of the network, E_L the learning error, and η the accuracy parameter corresponding to the probability (note the similarity with the regularisation equation (4.5)). The confidence interval is given by:

$$C\left(\frac{P}{\mathcal{VC}_{dim}}, E_L(\mathbf{W}), \eta\right) = 2\Psi\left(\frac{P}{\mathcal{VC}_{dim}}, \eta\right)\left(1 + \sqrt{1 + \frac{E_L}{\Psi(\frac{P}{\mathcal{VC}_{dim}}, \eta)}}\right) \tag{4.8}$$

where

$$\Psi\left(\frac{P}{\mathcal{VC}_{dim}}, \eta\right) = \frac{1}{P}\left((\ln\frac{2P}{\mathcal{VC}_{dim}} + 1)\mathcal{VC}_{dim} - \ln\eta\right) = \epsilon^2 \tag{4.9}$$

is essentially a function of the ratio $\frac{P}{\mathcal{VC}_{dim}}$.

The connection between generalisation error, learning error, confidence interval and number of patterns in the data set is illustrated in figure 4.5.

The only unknown parameter at this point is the capacity of the network expressed by $\mathcal{VC}_{dim}$. In equation (4.7) it was shown that the generalisation error will always exceed the learning error. It will, however, be possible to make the difference between these two quantities arbitrarily small if it is possible to make the ratio between P and $\mathcal{VC}_{dim}$ sufficiently large. The size of P, the data in the training set, will normally be limited by the problem domain or by the supervisor in order to learn something in a reasonable time. For a fixed number P, the learning error will decrease monotonically as the capacity $\mathcal{VC}_{dim}$ increases. Unfortunately, the confidence interval of equation (4.8) is a monotonically increasing function of $\mathcal{VC}_{dim}$ at a fixed P. This indicates that there will be an optimal $\mathcal{VC}_{dim}^{opt}$ where the generalisation error will have a minimum. For a $\mathcal{VC}_{dim}$ smaller than the optimal $\mathcal{VC}_{dim}$ the resources of the network will not be sufficient for learning the training set. Although the confidence interval is small, indicating a good correlation between the

[5] In [Vapnik 82]and [Blumer *et al.* 86] there is an alternative and often used exposition of inequality (4.7) given by:

$$P(\sup_{\mathbf{w}\in\mathbf{W}} \frac{E_G(\mathbf{W}) - E_L(\mathbf{W})}{\sqrt{E_G(\mathbf{W})}} > \epsilon) < 8 \triangle (2P) exp(-\epsilon^2 P/4).$$

where $\triangle(\cdot)$ is the *growth function*.

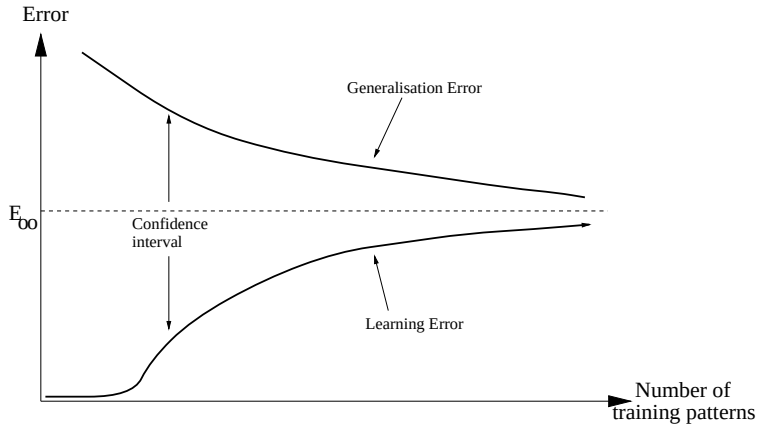


Figure 4.5: Illustration of the connection between generalisation error, learning error, confidence interval and number of patterns in the data set.

learning error and the generalisation error, this is no good while the learning error itself is large. For $\mathcal{VC}_{dim}$ bigger than the optimal $\mathcal{VC}_{dim}^{opt}$ there will be sufficient resources in the network to make the learning error small. But the confidence interval will be large and the learning error will no longer be a good estimate for the generalisation error. This corresponds to the over-fitting case in figure 4.1. The connection between generalisation error, learning error, confidence interval and capacity is illustrated in figure 4.6.

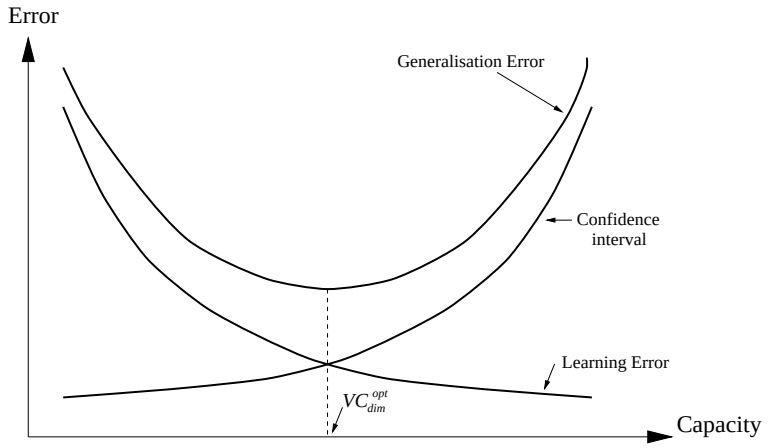


Figure 4.6: Illustration of the connection between generalisation error, learning error, confidence interval and capacity.

If an optimal capacity $\mathcal{VC}_{dim}^{opt}$ of a given network could be found, the number of units and the number of weights and thresholds would be known. So far nothing is said about the values of these parameters. It seems likely that there will be several weight constellations $\mathbf{w}_i$ from $\mathbf{w}$ that could implement the desired function. This was confirmed by Denker et al.(1988) [Denker *et al.* 87] in what they call a perturbation analysis. They took a well working network ($E_L = 0$), and perturbed the network, moving the weights to a new point in the weight space, and re-trained it. They found that the network was quite able to re-solve the task, returning to $E_L = 0$, but did not do so by undoing the

perturbation. In fact, it moved in some other direction and settled on a new point $\mathbf{w}_i$ in the weight space.

4.5.2 Probably Approximately Correct Learning Theory

The $PAC^{\heartsuit}$ learning theory was introduced by Valiant [Valiant 84] in 1984, where he defines a notation of learnability. The intention of the PAC theory is that successful learning of a function f^d taken from a class of functions $\mathcal{F}$ should, with high **probability**, be learnt by a function $f_{\mathbf{w}}$ (network) so that it is a good **approximation** of f^d. Hence the name *Probably Approximately Correct*. To be more precise, the PAC theory defines the concept *uniformly learnable* in the following way:

> Given enough training patterns it should be possible to find a function $f_{\mathbf{w}}$ (network) that with arbitrarily small error, $0 < \epsilon < 1$ is able to learn an unknown function f^d from a class of functions $\mathcal{F}$, with arbitrarily high probability $(1-\eta)$ where $0 < \eta < 1$, no matter which function from $\mathcal{F}$ that should be learned. The bounds to the training set size must be independent of the underlying distribution $P(\mathbf{V}_0, \mathbf{T})$.

Applying the definitions from paragraph 4.3 and choosing E as the indicator function, the following equation for *uniform learnability* appears:

$$P(\sup_{\mathbf{w}\in\mathbf{W}} |E(f^d, f_{\mathbf{w}})| > \epsilon) < \eta \tag{4.10}$$

The PAC theory also defines the concept *sample complexity*, which says:

> The smallest training set size, for which all functions in the function class are uniformly learnable with the learning function f_w, is called the *sample complexity* of f_w.

Valiant's work was defined in very broad terms and restricted to the Boolean functions ($B^N \rightarrow B$ where $B = \{-1, 1\}$). It was later extended to learning functions, taken from a class of functions defined by regions in the Euclidean n-dimensional space by [Blumer *et al.* 86]. They also applied the definition of $\mathcal{VC}_{dim}$ to prove the following interesting results:

> The class of unknown function $\mathcal{F}$ is uniformly learnable if and only if the $\mathcal{VC}_{dim}$ for $\mathcal{F} < \infty$
>
> For $0 < \epsilon < 1$ and a training set size at least:
>
> $$|P| \geq \max\left(\frac{4}{\epsilon}\log_2\left(\frac{2}{\eta}\right),\ \frac{8\mathcal{VC}_{dim}}{\epsilon}\log_2\left(\frac{13}{\epsilon}\right)\right) \tag{4.11}$$
>
> any binary function will be a learning function for $\mathcal{F}$.
>
> For $0 < \epsilon < 1/8, \eta < 1/100$ and a training set size less than:
>
> $$|P| < \max\left(\frac{(1-\epsilon)}{\epsilon}\, ln\left(\frac{1}{\eta}\right),\ \frac{\mathcal{VC}_{dim} - 1}{32\epsilon}\right) \tag{4.12}$$
>
> no function will be a learning function for $\mathcal{F}$.

This shows a significant gap in the inherent sampled complexity for learning functions: either the $\mathcal{VC}_{dim}$ is finite, and the true function is uniformly learnable with a training set at size $O(\frac{1}{\epsilon}\log_2(\frac{1}{\epsilon\eta}))$, or it is not uniformly learnable at all. In other words if the capacity is infinite the generalisation ability will be poor, recall the example in paragraph 4.2.1. with $f_W(U_L) = sign(sin(w_1V_{01} + w_0V_{00}))$.

These statements are surprisingly clear, because besides the numbers ϵ and η (which are chosen as needed or desired), the only unknown term is the $\mathcal{VC}_{dim}$. But as shown in the next paragraph, the $\mathcal{VC}_{dim}$ is unknown for most classes of functions.

4.6 Bounds to the *VC-dimension*

In general the $\mathcal{VC}_{dim}$ is unknown for most neural networks. An exception is the linear classifier as it was shown in paragraph 4.2.1. In this paragraph it will be shown, by making the assumption that a network only contains units with threshold transfer functions allowing one to use the result from the linear classifier, that it will be possible approximate the $\mathcal{VC}_{dim}$ for several networks.

4.6.1 $\mathcal{VC}_{dim}$ for the Linear Classifier

In the presentation of the $\mathcal{VC}_{dim}$ the linear classifier was used to illustrate the concept. A closer look at the illustrations in figure 4.2 and 4.3 reveals that the $\mathcal{VC}_{dim}$ for a linear classifier with a threshold transfer function in the output unit is simply $N_0 + 1$ (number of weights including thresholds), which is also equal to the dimension of the input data $\mathbf{V}_0^p = \{V_{01}^p, \ldots, V_{0N}^p\}+$ a fictive V_{00} to the threshold.

Combining this with the ideas from the paragraph on the *VC theory* it is possible to find a crude estimate of the number of patterns needed to obtain a good generalisation ability i.e. ϵ small. Assuming that the capacity of the network is so large that the network can be trained to classify all training patterns almost correctly, $E_L \approx 0$, $C(\frac{P}{\mathcal{VC}_{dim}}, E_L \approx 0, \eta)$, the confidence interval in equation 4.8 is reduced to:

$$C\left(\frac{P}{\mathcal{VC}_{dim}}, E_L \approx 0, \eta\right) = 4\frac{1}{P}\left((\ln\frac{2P}{\mathcal{VC}_{dim}} + 1)\mathcal{VC}_{dim} - \ln\eta\right) = 4\epsilon^2 \tag{4.13}$$

By manipulating and using this expression as both $\mathcal{VC}_{dim}$ and P become large the following crude approximation for P will be valid:

$$P > \frac{(N+1)\ln(N+1)}{\epsilon^2} \tag{4.14}$$

Thus the number of training examples needed to train a linear classifier and obtain a good generalisation in the worst case is inversely proportional to the second order of ϵ. Since ϵ^2 is equal 4 times the confidence interval it must be small in order to get a good generalisation ability, so the number of training patterns needed can be expected to be high.

In [Cover 65] it is further shown that if the transfer function is changed from a threshold function to a pulse transfer function (RBF unit) the $\mathcal{VC}_{dim}$ will be $N_0 + 2$ (number of weights including thresholds and radius).

4.6.2 $\mathcal{VC}_{dim}$ for a Two-Layer Feed-Forward Network

The exact $\mathcal{VC}_{dim}$ for an MLP is in general unknown, even in the simple case where the MLP only consists of a single hidden layer it is not known. Baum and Hausler (1989) found by using the results from the linear classifier[6] that for an MLP with N_1 hidden threshold units, $|\mathbf{W}|$ weights (including thresholds) and a single threshold output unit, the upper bound to the $\mathcal{VC}_{dim}$ is

$$\mathcal{VC}_{dim} \leq 2|\mathbf{W}| \log_2(\mathbf{e}N_1) \tag{4.15}$$

where $\mathbf{e}$ is the base of the natural logarithm.

Applying this bound of $\mathcal{VC}_{dim}$ in a combination with the ideas from the paragraph on the *VC theory* and the bound to the smallest training set size needed given in (4.12) on the *PAC theory*, Baum and Hausler showed that:

> for $\eta \leq 1/100$ and $0 < \epsilon \leq 1/8$, a network will be able to correctly classify a fraction $(1 - \epsilon)$ of unknown patterns, if this is drawn from the same distribution as the training patterns and it is possible to train the network with $P \geq O(\frac{\mathbf{W}}{\epsilon} \log_2(\frac{\mathbf{e}N_1}{\epsilon}))$ so at least a fraction $(1 - \frac{\epsilon}{2})$ of the training patterns is correctly classified.

As a corollary of the *PAC theory* they also found that for a network with N_0 inputs and one fully connected hidden layer with N_1 units, the lower bound of the $\mathcal{VC}_{dim}$ is $2\lfloor N_1/2 \rfloor N_0 \approx |\mathbf{W}|$ where $\lfloor x \rfloor$ denotes the largest integer not greater than x and $|X|$ the number of elements in X.

4.6.3 $\mathcal{VC}_{dim}$ for Networks Built by Construction Algorithms

There are several construction algorithms, see chapter 6, that are using the simple perceptron as their basis. They try to improve the performance by successively inserting units between the perceptron output and the inputs. In general these methods start by connecting a new unit to the input and train it, before it is connected to the output unit. The training of the new unit will normally continue until it has achieved a satisfying or best possible performance. The weights between the new unit and the input are then frozen (never changed) and the output from the new unit is connected to the output unit. A few examples are illustrated in figure 4.7.

[6] The estimation of the growth function $\triangle_{\mathcal{F}}(P)$ in [Baum and Haussler 89] is set equal to the product of all $\triangle_{\mathcal{F}_i}(P)$ from each hidden unit, $\triangle_{\mathcal{F}}(P) = \prod_{i=1}^{N_1} \triangle_{\mathcal{F}_i}(P)$. A sharper bound may be obtained by realising that the $\triangle_{\mathcal{F}_i}(P)$ is member of a multi set, because permutations amongst the $\triangle_{\mathcal{F}_i}(P)$ should not be counted.

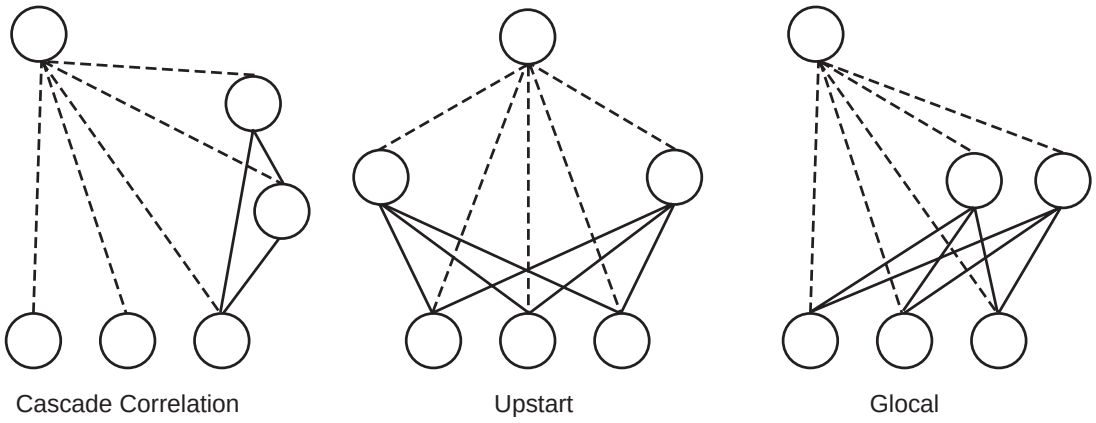


Figure 4.7: Examples of networks built by construction algorithms

The output unit still has all its original inputs so the number of inputs is increased by one for every new unit. There are no ways for the output units to distinguish one input from the other, it will just look as if the input dimension has increased. The situation is now exactly the same as for the linear classifier where it is known that the $\mathcal{VC}_{dim}$ is equal to the number of inputs plus threshold, $N_0 + 1$, therefore the $\mathcal{VC}_{dim}$ for the new network is:

$$\mathcal{VC}_{dim}^{New} = \mathcal{VC}_{dim} + N^{New} = N_0 + 1 + N^{New} \tag{4.16}$$

where N^{New} is the number of new units inserted.

Thus the number of training examples needed to train these networks and obtain a good generalisation is similar to the expression in equation (4.15) apart from the additional term N^{New}.

It should be stressed that the reason for this simple connection in equation (4.16) is that the number of different patterns presented as input to the output unit is still the same. The essential thing here is that the inserted units do not map any of the original input patterns into a new higher dimensional pattern where they are no longer unambiguous, if they were so in the first place. Consider the case with P distinct patterns of dimension N_0. If the dimension is increased by one, P patterns would still be distinct no matter which value the additional dimension may take.

4.7 Résumé and Open Questions

This chapter has shown both a practical and a theoretical approach to the concept of generalisation. A simple way to obtain information about this was given through the idea of measuring the generalisation error by splitting the available data into two separate sets, a training set and a test set. A theoretical analysis based on the use of the $\mathcal{VC}_{dim}$ was introduced and it showed that there are certain difficulties in determining a useful $\mathcal{VC}_{dim}$. The result can be summarised by the illustrations in figure 4.8.

The approximations applied in order to find a $\mathcal{VC}_{dim}$ that is mathematically manageable are commonly very crude and therefore the gap to real life applications is still large.

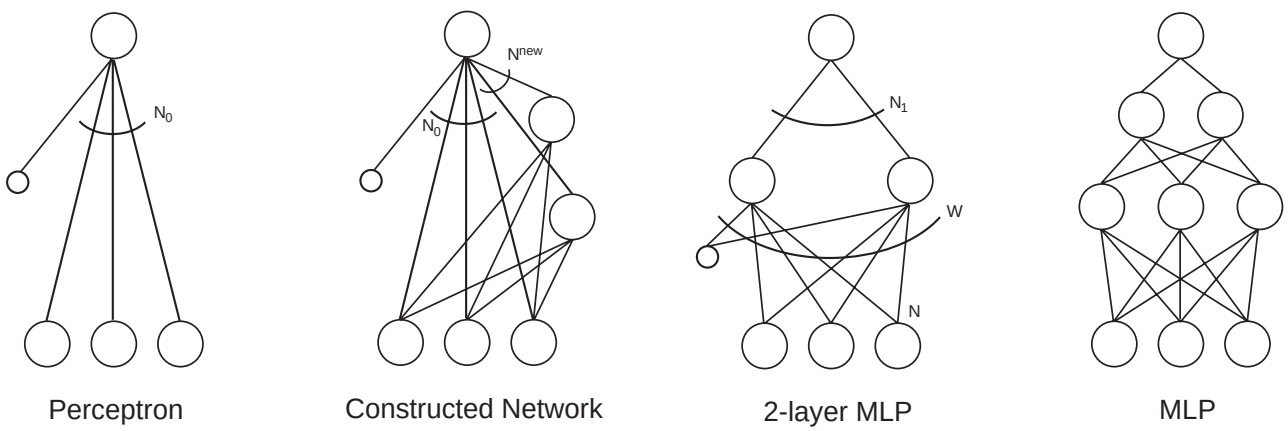


Figure 4.8: Networks and the corresponding $\mathcal{VC}_{dim}$

As the $\mathcal{VC}_{dim}$ is the key to both the VC theory and the PAC theory one may wonder if this is of any value. From a practical point of view, the main results from the mathematical analysis were:

> It confirmed the belief that there is a trade-off between the learning error and the optimal size of the architecture in order to achieve a good generalisation.
>
> The number of training patterns depends in general on ϵ, $0 < \epsilon \leq 1/8$, which determines the size of the difference between the training error and the error on unknown data.
>
> For an MLP with a single layer the size of the training set should be about 10 times the number of weights. [7]

Theoretically both the VC theory and the PAC theory offer a framework that seems to be able to gather scientists with different academic backgrounds. The field is slowly growing, but the big breakthrough is still to come. One of the newest ideas of making the VC theory more useful is an attempt to measure the $\mathcal{VC}_{dim}$ [Vapnik *et al.* 94] and [Bottou *et al.* 94] from experiments. So far without convincing results.

The overall conclusion drawn from the description in this chapter is that the neural network that is able to learn the training data in a satisfactory way using the smallest number of parameters, is the network with the largest probability of a good generalisation ability. This philosophy is often referred to as Ockham's Razor$^{\heartsuit}$.

[7] This seems to be plausible and agrees with an often used rule of thumb, called Uncle Bernie's rule in [Morgan and Bourlard 90] after Bernard Widrow [Widrow 87].

Chapter 5

Regularisation and Pruning

- Improving Predetermined Architectures

This chapter describes several methods to improve the generalisation ability of a neural network with a predetermined architecture containing many parameters. The methods can be split into 2 groups: one which tries to improve the generalisation ability during learning and another which tries to improve the generalisation ability of a well working network. The first group is called *regularisation* and includes methods known as *Weight Decay* and *Early Stopping*, while the latter is called *Pruning* and includes methods such as *Magnitude Based Pruning*, *Optimal Brain Damage* and *Optimal Brain Surgeon*. Results from experiments with these methods will be given.

5.1 Introduction

The various generalisation theories seem to agree that the model/machine/neural network that is able to learn the training data in a satisfactory way using the smallest number of parameters, is the model/machine/neural network with the largest probability of a good generalisation ability. This means e.g. that a network with many parameters that gives a low learning error should not be preferred to a network with few parameters giving a slightly larger, but still satisfactory, learning error. A widely used class of techniques starts by choosing a network with a high capacity, i.e. many weights and units so that a small learning error can be expected (over-fitting). These techniques then try to improve the generalisation ability either by adding an additional term to the error function used in training, using a validation set to decide when to stop training, or by removing redundant parameters (weights or/and units) from a well trained network based on some measurement of *saliency*. The first method can be viewed as a variation of a well established statistical technique known as regularisation. The second method of stopping training can also be interpreted as a form of *regularisation* but in fact it seems to be one of the true innovations resulting from neural network research. The last method can be viewed as a more established technique called *specification search* [Leamer 79].

5.2 Regularisation

Regularisation comes from statistical mathematics and is based on the idea that the generalisation error is a sum of two components: the error on the training set and an additional term depending on the network's complexity. The generalisation error $E_G(\mathbf{W})$ is approximated in the following way:

$$E_G(\mathbf{W}) \approx \sum_{p=1}^{P} E(\mathbf{V}_L^p, \mathbf{T}^p) + \lambda C(f_{\mathbf{w}}) \tag{5.1}$$

where $C(f_{\mathbf{w}})$ is a monotonically increasing cost-function of the mapping function $f_{\mathbf{w}}$ as the complexity of $f_{\mathbf{w}}$ increases[1]. λ is the regularisation parameter that controls the compromise between the degree of smoothness of solution and its closeness to the data. The value of λ depends on the problem and on the choice of the cost-function $C(f_{\mathbf{w}})$. A way to control the smoothness of a neural network is to constrain the value of the parameters or the number of parameters.

5.2.1 Weight Decay

Weight Decay$^{\heartsuit}$ methods are based on the regularisation technique where the additional term tries to penalise weights with large values. The simplest way of weight decay is to let the additional term be a sum of square weight values, so the error function becomes:

$$E_G(\mathbf{W}) \approx \sum_{p=1}^{P} E(\mathbf{V}_L^p, \mathbf{T}^p) + \frac{1}{2}\lambda \sum_{l=0}^{L} \sum_{k=0}^{N_l} w_{lk}^2 \tag{5.2}$$

where l and k are indexes to all weights in the network.

Applying the Gradient Descent method to this error function, the update rule for the weights will then be given by:

$$\triangle \mathbf{w} = -\eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} - \eta\lambda\mathbf{w} \tag{5.3}$$

Straight forward manipulation of this equation shows that this is equivalent to performing an ordinary weight update and then reducing the value of the weight by a small term $\eta\lambda$ so the new value of the weight becomes:

$$w_{ij}(t+1) = (1 - \eta\lambda) w_{ij}(t) \tag{5.4}$$

where $w_{ij}(t)$ is the value of the weight after a normal weight update. As this equation shows weights that are not continually updated by the Gradient Descent will gradually decay, hence the name Weight Decay.

A drawback to this simple Weight Decay method is that the decay rate is equal for all weights. [Hertz *et al.* 91] gives a review of methods very similar to the simple Weight Decay that try to use different decay rates, so e.g. small weights will decay more rapidly.

[1] Note the similarity with equation (4.7) in the paragraph on the VC theory.

The idea of constraining the value of the weights can - from a Bayesian view - be interpreted as applying *a priori* knowledge about the best solution. For networks with sigmoid transfer functions, the use of simple Weight Decay is equivalent to incorporation of prior knowledge of the best possible choice of network is a network working as a linear model (small weights make the units work in their linear areas). In [MacKay 92], [MacKay 91] and [Buntine and Weigend 91b] a Bayesian approach was used to come up with various rules for changing the weights, including changing the decay rate dynamically during learning.

5.2.2 Early Stopping

Another way of constraining the parameters is to stop the training before the network has made use of many of its degrees of freedom and over-fitting appears. This idea is called Early Stopping$^{\heartsuit}$ and belongs to the class of cross-validation techniques, which basically means that the data set is split into two sets, a training set and a test set, see chapter 4. In the Early Stopping method the training set is further split into two sets, a reduced training set and a **validation set**. The training set is used in the usual way while the validation set is used for test during learning. The Early Stopping is then performed by observing the error on the validation set and then stopping the training as soon as the error starts to increase. Figure 5.1 illustrates the effect of Early Stopping.

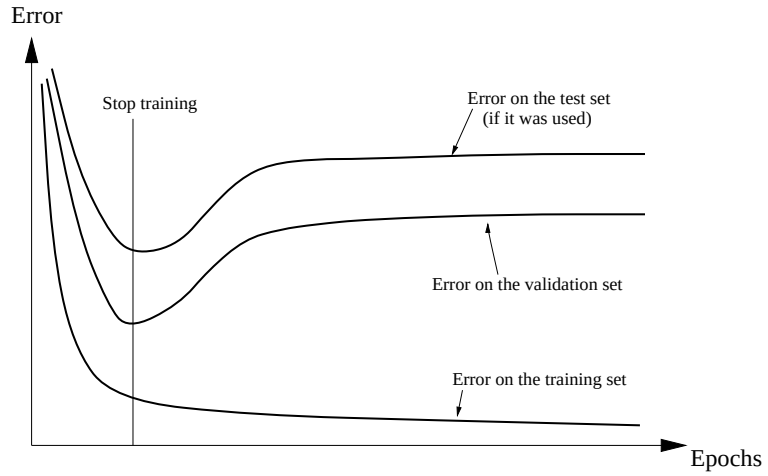


Figure 5.1: Illustration of the Early Stopping principle

In practice the actual procedure is a bit more complicated since there may be many local minima in the validation set error curve and since in order to recognise a minimum one has to train until the error rises again, so that resetting the network to an earlier state is needed in order to actually stop at the minimum. One simple way of handling this situation is to keep track of the smallest error on the validation set and then stop the training as soon as the fraction between the current error on the validation set and the smallest error exceeds a certain threshold.

Early Stopping has inhereted the advantages from Cross-Validation: It is easy to use, the user need not know anything about the distribution of the data, and it allows the use of two error functions, one for learning and one for test, see chapter 4. In addition Early

Stopping also allows the use of two error functions in the learning phase, because the purpose of the validation set in the learning phase is similar to the purpose of the test set in the test phase, which is to measure the number of correct classifications. It should be stressed that the validation set has nothing to do with the test set, except that they are assumed to be drawn from the same distribution of data, so the network's performance on the validation set is expected to be similar to the performance on the test set. Early Stopping has, unfortunately, also inhereted the drawbacks from Cross-Validation, as there is no way of telling how to split the data set in such a way that the result will be reliable.

5.3 Pruning

The main idea behind all pruning$^{\heartsuit}$ methods is to keep the learning error $E_L(W)$ for a well working feed-forward network as low as possible and at the same time to reduce the complexity i.e. number of weights and sometimes units. The number of different pruning methods is gradually increasing. A survey of some of the earlier pruning methods can be found in [Reed 93]. In the following paragraphs three popular pruning methods will be described.

5.3.1 Magnitude Based Pruning

One of the simplest methods for reducing the complexity of a neural network is called Magnitude Based pruning (MAG). The method is based on the idea of eliminating the weakest connection, i.e. the weight with the smallest magnitude. For the simplest MAG version the algorithm is:

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Store the weights
4. Find the weight with the smallest magnitude and eliminate it
5. Retrain the network
6. If a certain stop criterion is fulfilled go to step 7 else go to step 3
7. Restore the weights saved in step 3 and stop

There are several more or less sophisticated versions of this method. If a network contains many weights, it will normally be possible to speed this procedure up by allowing elimination of several weights before retraining. A sophisticated version of this simple idea is to judge the magnitude ratio among all weights in the group of weights to each single unit. If the ratio between the largest weight in the group and the sum of the smallest weights in the same group is bigger than some threshold, the group of the smallest weights may be eliminated. If no groups are removed the original MAG should take over.

5.3.2 Optimal Brain Damage

A more mathematical approach using information from the second order derivatives of the error function to perform pruning has been derived by LeCun in a method called Optimal Brain Damage (OBD) [Le Cun *et al.* 90]. It is based on the use of the Taylor expansion to express an estimate of how the training error will change as the weights are perturbed:

$$\delta E_T \approx \sum_i \mathbf{G}_i \delta \mathbf{w}_i + \sum_i \frac{1}{2} \delta \mathbf{H}_{ii} \delta \mathbf{w}_i^2 + \sum_{i \neq j} \frac{1}{2} \mathbf{H}_{ij} \delta \mathbf{w}_i \delta \mathbf{w}_j + (||\mathbf{W}||^3) \tag{5.5}$$

where $\mathbf{G}$ is the gradient $\frac{\partial E_L}{\partial w_i}$, $\mathbf{H}$ is the Hessian $\frac{\partial^2 E_L}{\partial w_i \partial w_j}$ and the $(||\mathbf{W}||^3)$ term is the remainder.

The method makes the following assumptions:

the network is trained to a point where the gradient is zero so the first term in the equation can be neglected and the "quadratic" approximation assumed that the cost function is nearly quadratic also holds so that the last term in the equation can be neglected.

δE caused by deleting several parameters is the sum of the δE's caused by deleting each parameter individually so the off-diagonal part of the second order term is zero.

The equation is then reduced to

$$\delta E_L = \frac{1}{2} \delta \mathbf{H}_{ii} \delta \mathbf{w}_i^2 \tag{5.6}$$

The δE_L term is called *saliency* and expresses the change in the cost-function due to the elimination of weights.

The algorithm is:

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Store the weights
4. Compute the second derivatives $\mathbf{H}_{ii}$ for each parameter
5. Compute the saliencies for each parameter: $\delta E_L = \frac{1}{2} \delta \mathbf{H}_{ii} \delta \mathbf{w}_i^2$
6. Delete the weight with the smallest saliency
7. Retrain the network
8. If a certain stop criterion is fulfilled go to step 9 else go to step 3
9. Restore the weights saved in step 3 and stop

5.3.3 Optimal Brain Surgeon

The Optimal Brain Surgeon (OBS) [Hassibi and Stork 93] can be considered as an extension of OBD. It is also based on the Taylor expansion, but contrary to OBD it does not assume that the off-diagonal of the Hessian is zero. Instead it reformulates the goal. The elimination of $\mathbf{w}_j$ can be expressed as: $\delta\mathbf{w}_j + \mathbf{w}_j = 0$ or $\mathbf{e}_j^T\delta\mathbf{w} + \mathbf{w}_j = 0$ where $\mathbf{e}_j$ is the unit vector in the weight space corresponding to (scalar) weight $\mathbf{w}_j$. The goal is then to solve:

$$\min_j \left(\min_{\delta\mathbf{w}} (\frac{1}{2}\delta\mathbf{w}_j^T\mathbf{H}\delta\mathbf{w}_j) \right) \quad ; \text{ so that } \quad \mathbf{e}_j^T\delta\mathbf{w} + \mathbf{w}_j = 0 \tag{5.7}$$

or expressed in terms of Lagrange Multipliers:

$$\delta E_L = \frac{1}{2}\delta\mathbf{w}_j^T\mathbf{H}\delta\mathbf{w}_j + \lambda(\mathbf{e}_j^T\delta\mathbf{w} + \mathbf{w}_j) \tag{5.8}$$

By taking functional derivatives the following equations appear:

$$\delta\mathbf{w} = (\frac{\mathbf{w}_j}{\mathbf{H}_{jj}^{-1}})\mathbf{H}^{-1}\delta\mathbf{e}_j^T \quad \text{and} \quad \delta E_L = \frac{1}{2}(\frac{\mathbf{w}_j^2}{\mathbf{H}_{jj}^{-1}}) \tag{5.9}$$

The δE_L term is again called *saliency* and expresses the change in the cost-function due to the elimination of weights. The $\delta\mathbf{w}$ indicates how all weights should be adjusted, according to the elimination of a weight. This means that the network does not demand retraining. The only "learning" parameter involved is an α which comes from initialising an iterative method that together with the Sherman-Morrison formula (see e.g. [Golub and Loan 83]) are employed to calculate the Inverse Hessian matrix in the following way:

$$\mathbf{H}_{p+1}^{-1} = \mathbf{H}_p^{-1} - \frac{\mathbf{H}_p^{-1}\mathbf{X}^{(p+1)}\mathbf{X}^{(p+1)T}\mathbf{H}_p^{-1}}{P + \mathbf{X}^{(p+1)T}\mathbf{H}_p^{-1}\mathbf{X}^{(p+1)}} \quad \text{with} \quad \mathbf{H}_0^{-1} = \alpha^{-1}\mathbf{I} \quad \text{and} \quad \mathbf{H}_P^{-1} = \mathbf{H}^{-1} \tag{5.10}$$

where P is the number of patterns and $\mathbf{X}$ is a vector containing information of the second derivative, see [Hassibi and Stork 93] for a detailed description.

The OBS algorithm is:

1. Choose a reasonable network architecture
2. Train the network until a reasonable solution is obtained
3. Store the weights
4. Compute the inversion of the second derivatives $\mathbf{H}^{-1}$
5. Compute the saliencies: $\delta E_L = \frac{1}{2}(\frac{\mathbf{w}_j^2}{\mathbf{H}_{jj}^{-1}})$ for each parameter that has not been eliminated
6. Find the weight j with smallest saliency and delete it

7. Use j from step 6 to update all weights: $\delta \mathbf{w} = (\frac{\mathbf{w}_j}{\mathbf{H}^{-1}_{jj}})\mathbf{H}^{-1}\delta \mathbf{e}_j^T$

8. If a certain stop criterion is fulfilled go to step 9 else go to step 3

9. Restore the weights saved in step 3 and stop

5.3.4 Tests and Experiments

In chapter 4 it was justified that pruning will improve the generalisation ability. It was also shown that there could be many possible weight constellations (points in the weight space) that would yield equally valid generalisations for an optimal or nearly optimal capacity. Although OBD and OBS have established a mathematical foundation to get a good estimate of the saliency, they still do not make a quantitative statement of how well they improve the generalisation ability. The same is also true for magnitude based methods. So it seems there is no theoretical evidence that one method will improve the generalisation more than the other.

In order to get an idea of which method is the best the three methods were tested on the benchmark MONK's problems described in a report made by [Thrun *et al.* 91]. In this report they also described 3 fully connected neural networks trained by a Back-Propagation with Weight Decay (BPWD) that outperformed all other approaches (network and rule-based systems) on these problems in an extensive machine learning competition.

The goal here was to find how many weights could be eliminated by the different methods while still performing as well as [Thrun *et al.* 91]. The result from these experiments is shown in table 5.1 and the architecture which the different methods produce on the MONK 1 problem is shown in figure 5.2.

	BPWD	MAG	OBD	OBS
MONK 1	58	13	13	14
MONK 2	39	15	26	15
MONK 3	39	6	9	4

Table 5.1: Number of weights needed for MAG, OBD and OBS to make the same performance as the Back-Propagation with Weight Decay (BPWD) found by Thrun et al.(1991), on the MONK's problems.

In the training and retraining phases of both MAG and OBD simple Weight Decay was applied, the result was significantly better than without Weight Decay. A more detailed description of the learning parameters applied in this test and the weight pruning sequences can be found in Appendix C.

Many similar tests were made on the MONK's problems given new start conditions. The new start conditions were created by training the original network from random weights so the weight-start-position of the test network would differ from test to test. The result was the same, all methods were capable of reducing the number of weights, and none of the methods was the best every time. Adjusting the parameters in the

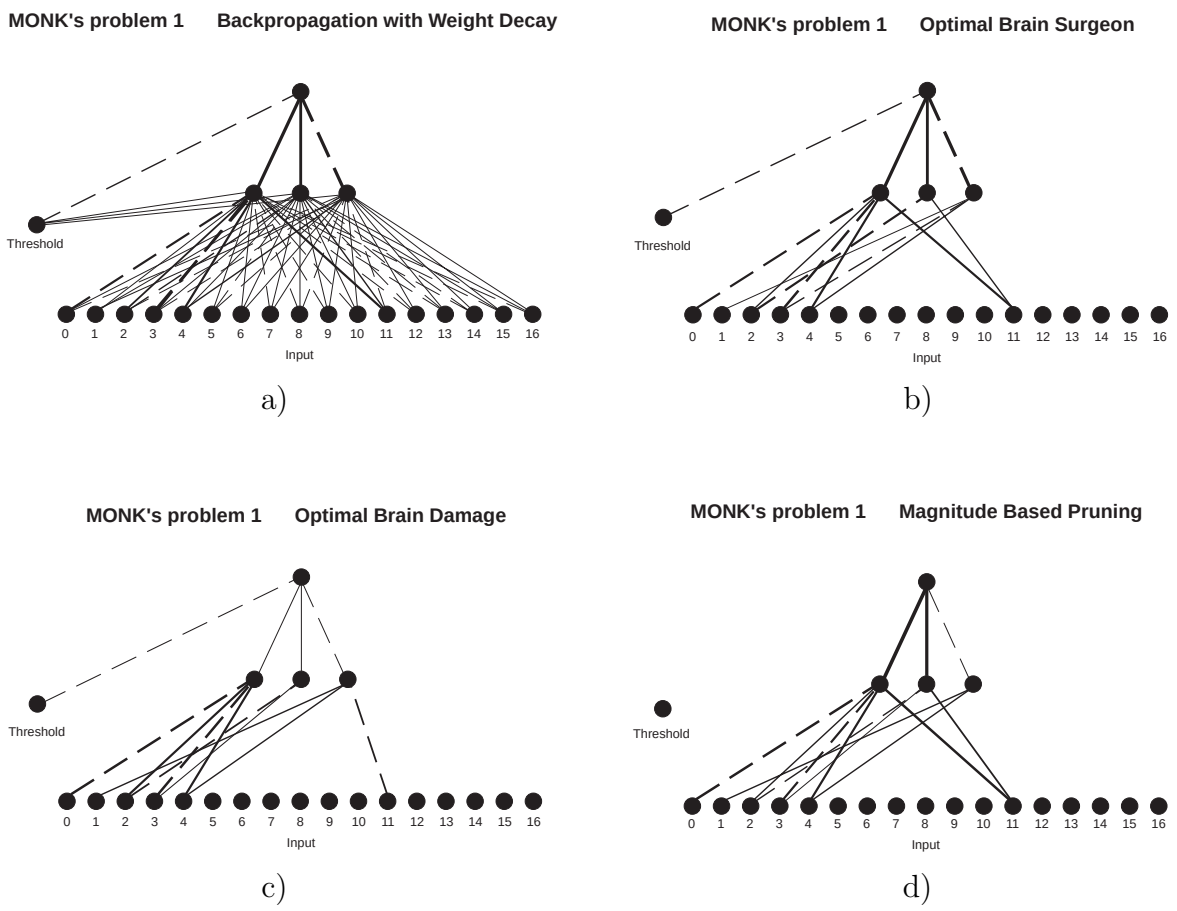


Figure 5.2: Architectures produced by the different methods on the MONK's problem 1. Solid lines denote excitatory connections and dashed lines inhibitory. a) The original network found in Thrun et al. 1991. b) The original network pruned with OBS. c) The original network pruned with OBD. d) The original network pruned with MAG.

learning algorithm, MAG and OBD performed as well as OBS and sometimes better. Further experiments with similar results can be found in [Knudsen and Vlk 94].

5.3.5 Remarks

These experiments confirmed that no single method was the best all the time and that there were several optimal solutions to the same problem. The experiments did, however, show that OBS was by far the most stable and robust method. If it was not the best it was always close to the best.

Hassibi and Stork (1992) have shown through an example of a 5-node solution to the XOR problem that only OBS will always be able to remove the right weight while both OBD and especially MAG often fail to do so. The reason why OBS works perfectly here is that the minimum error at the start is equal to the global minimum.

This result can be explained by the fact that OBS will stay close to the local minimum from where it begins the pruning. It tries to remove one weight and then projects the

error surface to a space one dimension smaller than the previous, in such a way that it will stay at the same local minimum error in the surface. This will work very well as long as the local minimum from where OBS starts is close to the global minimum. But if the original (start) minimum error is not the smallest local error, OBS will never find this point because it does not retrain. The method is therefore stable but may also fail to find a new local minimum.

Both OBD and MAG will be able to find such a new local minimum depending on their learning algorithm. The general problem with the strategies used by MAG and OBD is that they are one step predictors, which means that they take one "optimal" step, recalculate the conditions and take another "optimal" step. There is no guarantee that these two steps together are optimal. How optimal the steps are all together will depend on the recalculations of the conditions which are directly dependent on the learning algorithm and its parameters. Therefore these methods are unstable.

5.4 Résumé and Open Questions

This chapter has shown some very popular methods to improve the generalisation ability of a neural network with a predetermined architecture containing many parameters.

The difference between regularisation and pruning is that pruning is done in two steps: the first step is training of the network to a minimum and then the reduction of parameters is done in a second step which might, however, include retraining. The question still remains open whether one could benefit from combining the different methods. The experiment showed that Weight Decay along with pruning methods that require training and retraining is a good idea. Several successful tests are shown in [Finnoff *et al.* 93] where a combination of Early Stopping is used in connection with various regularisation and pruning methods, including Weight Decay and OBD.

The experiments described in this chapter showed that no single method was the best all the time but that OBS was by far the most stable and robust. To choose among these different methods, memory requirements, computing time, etc. should be taken into consideration. For real life applications where the complexity might be very large, the following pruning scheme seems reasonable. Start with MAG until the network has a size that will allow the use of OBS, and then make an effort to find a "good" local minimum in order to give OBS good start conditions. When such a minimum is found OBS should be used to do the rest of the pruning. When OBS stops the network should be retrained in order to get $\delta E = 0$ and OBS should be started again.

One of the open questions in connection with all pruning methods is when to stop. Different stop criteria may be used, it is e.g. often seen that the increase in the learning error after retraining is used or as far as OBS is concerned the size of the saliency (expected increase in error) is used. An obvious way is instead to employ a validation set and stop the pruning when the validation ratio exceeds a threshold as described in the paragraph on Early Stopping.

The general conclusion is that it does not seem to be important which method is used for pruning as long as one method is used. There is no doubt that the development of new regularisation terms, weight decay terms and pruning methods will continue for some years, but until a measurement of the generalisation ability is found there is no quality measurement for preferring one method to the other, except for practical reasons.

Chapter 6

Construction Algorithms

This chapter deals with construction algorithms which is a general term describing methods that successively build neural networks. Only construction algorithms employing supervised learning algorithms are considered here. These algorithms can be split into two classes: one where the transfer function of the hidden units is either a threshold or a sigmoid function and another where each hidden unit's transfer function is a Radial Basis Function[1]. Several well-known construction methods/algorithms from each class including the *Tower Construction Algorithm*, *Inverted Pyramid Construction Algorithm*, *Distributed Construction Algorithm*, *Sequential Learning Algorithm*, *Cascade-Correlation Algorithm*, *Restricted Coulomb Energy Algorithm* and *Resource-allocation network Algorithm* are described by algorithms and graphical illustrations. All these algorithms have served as inspiration to a new construction algorithm that combines both types of units. This algorithm is called *the GLOCAL Algorithm* and is derived and described in detail at the end of this chapter.

6.1 Introduction

In chapter 4 it was shown that the architecture of a neural network is important for the performance although it was not clearly stated how the architecture should be in order to achieve a good performance. Instead of starting with a network with too many parameters and then constraining the parameters or removing some like e.g. the pruning methods in chapter 5, it is more attractive to start with a small network and gradually increase the number of parameters to an appropriate size. In the last decade several methods for successively building a neural network during training have been proposed. These methods are often called *growth* or *constructive* and sometimes "start small and add". A survey of some constructive algorithms can be found in [1].

Gallant was one of the first to come up with several supervised learning methods for dynamically building and training neural networks using units with threshold transfer functions [Gallant 86]. Many of the later derived construction algorithms$^{\heartsuit}$ may be interpreted as extensions of Gallant's ideas. The more popular methods derived from Gallant's algorithms include algorithms such as the Upstart algorithm [Frean 90] and the Tiling algorithm [Mezard *et al.* 89].

[1]Before reading this chapter it might be a good idea to take a second look at the different transfer functions in chapter 2.

The class of networks using RBF units in the hidden layer have for several years shown good performances and are well suited for real life applications implemented in hardware, see RCE [Reilly *et al.* 82] and RAN [8]. This class can be dated back to at least 1982 where the Restricted Coulomb Energy [2] (RCE) network was introduced as a neural model for Category Learning. It was at that time a novelty distinctive enough from other models to be awarded a patent by the US Patent Office. Today it seems to be one of the really successful networks, as it is the heart of a learning system from the American company Nestor Inc., formed by the three inventors of the method, D.L Reilly, L.N. Cooper and C. Elbaum [Cooper *et al.* 88]. Many of the more recent construction algorithms using RBF units are very similar to the RCE network/methods as they are also based on the idea of category (cluster) learning, see e.g. [1] and [4].

The motivation for looking at construction algorithms is that they have several advantages compared to networks with a given architecture, e.g.:

- *Speed:* The overall development time of a neural network is faster than the classical approach i.e. where the design is basically done on a qualified and academic guess and some rules of thumb, that are repeated until success. Learning is often also faster while many construction algorithms are trained in modules.
- *Scaling properties:* Large network models and more training data can be handled.
- *Analytic tractability:* Analytic bounds to scaling and generalisation ability can be derived in many cases, thanks to the simplicity and modular structure of the underlying model, see chapter 4.

6.2 Simple Construction Algorithms

Three of the most simple construction algorithms were proposed by Gallant in [Gallant 86]. The following description of these methods serves as an introduction to the field because they are easy to understand and contain many of the terms and ideas which are common to many of the later algorithms in the field.

The first method known as the *Tower Construction Algorithm* starts by training a simple perceptron as well as possible. If the result is not satisfactory all weights to the perceptron will be **frozen**, which means that their values will never be changed during the learning phase. A new output unit is then inserted and connected to all input units and the previous output unit. The procedure is then repeated so all weights to the new output unit are trained and if the result is still not satisfactory, the weights are frozen and a new output unit is inserted. This continues until the result is satisfactory or a predetermined number of iterations has been executed. Figure 6.1 illustrates the Tower Construction learning algorithm and its corresponding architecture.

It should be noted that since the weights to all the previous output units are frozen when the current output unit is trained, the situation is always the same as for a simple

[2] The error function is often called an energy function by physicists and the hidden units can be viewed as a number of limited Coulomb potentials, hence the name RCE. Funny though that the inventors' names are **R**eilly, **C**ooper and **E**lbaum.

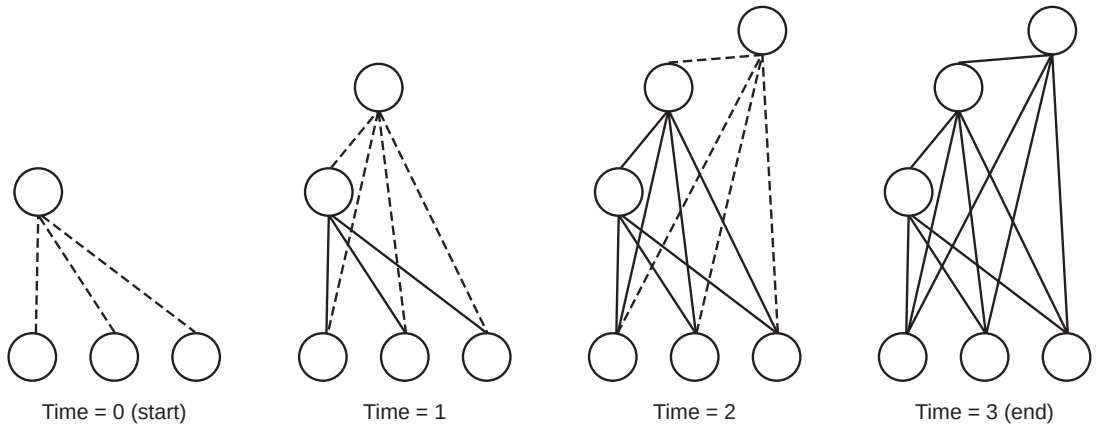


Figure 6.1: Evolution of the Tower Construction Algorithm. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution.

perceptron except that the number of inputs increases. This idea of just training a single unit at a time is typical for most construction algorithms. For a network with threshold units it allows the use of a simple learning method like the Pocket Algorithm. For networks with units employing continuous transfer functions it is also a good idea, because it makes the training of the network very quick (see the paragraph below on CCA).

The second method proposed by Gallant is called the Inverted Pyramid Construction. It is similar to the Tower Algorithm except that when a new output unit is inserted it is connected to all previous output units in contrast to the Tower Algorithm where it was only connected to the previous output unit in the layer below. The principle of the Inverted Pyramid Construction algorithm and its corresponding architecture are shown in figure 6.2.

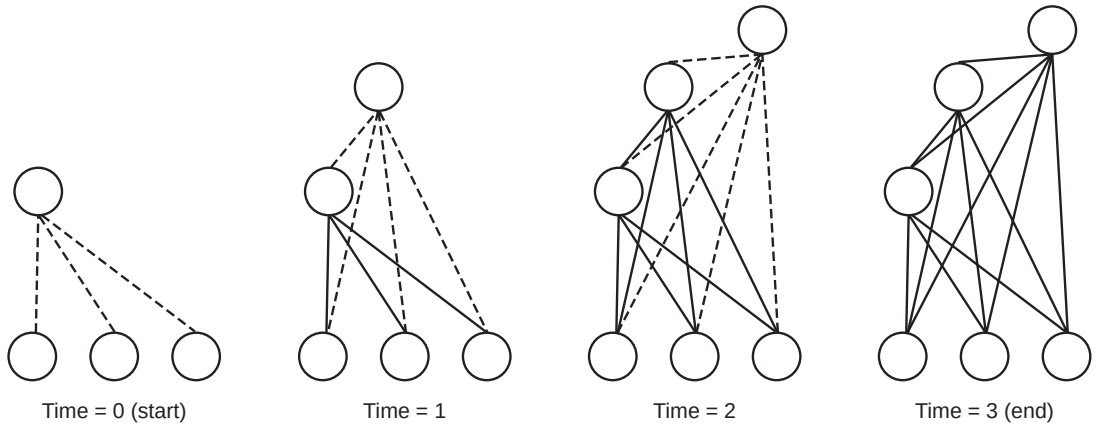


Figure 6.2: Evolution of the Inverse Pyramid Construction Algorithm. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution.

The following generic pseudo-algorithm describes both the Tower Construction and Inverted Pyramid Construction algorithms:

1. **Initial configuration**: The algorithm begins with a simple perceptron with N input units and a single output unit. N is given on the basis of the problem that the network is to learn. The maximum number of iterations is set to say I_{Max} and a counter i is set to 1.

2. **Initial training**: The perceptron is trained on the entire training set $\{(\mathbf{V}^p, \mathbf{T}^p) \mid p = 1, \ldots, P\}$, until the performance of the network is as good as possible. If the error on the training set is satisfactory go to step 5.

3. **Freeze and insert**: All weights to the current output unit are frozen and a new output unit is inserted into the network. The new output unit is as regards the **Tower Construction** connected to all input units and the previously inserted output unit and as regards the **Inverse Pyramid Construction** connected to all input units and all previously inserted units.

4. **Retraining the network**: All the incoming weights to the new output units are randomly initialised and the new perceptron is trained on all patterns. i is incremented by one. If i is smaller than I_{Max} and the error on the training set is smaller than before the training, go to step 3. Otherwise: go to step 5.

5. **Stop**: The algorithm stops.

The goal of these two methods is to make the error on the training set become as low as possible. Gallant uses his own *Pocket Algorithm*$^\heartsuit$ [Gallant 86] for training. The Pocket Algorithm has two nice properties: if the problem is linearly separable it will eventually be able to classify all patterns correctly; if the problem is not linearly separable it will in finite time find a solution where the output unit will produce a correct output on as many input patterns as possible. Both the Tower Construction and Inverted Pyramid Construction algorithms have inherited these properties.

The third method proposed by Gallant is called the Distributed method. The idea is to start with a two layer network with a single output unit and several hidden units with randomly chosen and frozen weights. If - after training the connections to the output unit - the performance of the network is not as desired, new hidden units with randomly chosen and frozen weights are added. A newer version allowing short cuts (connections between input and output) was proposed in [Gallant 90]. The algorithm is thus changed and will start with a simple perceptron. If the performance of the perceptron after training is not as desired, new hidden units with randomly chosen and frozen weights are added. Gallant argues that such units would be feature detectors and using enough of these units the task will eventually be solved. In theory it can be shown that this is true (see [Minsky and Papert 69] for the boolean case and [Hornik *et al.* 89] for the general case), but it may unfortunately require an exponential large number of hidden units. Gallant does not specify the features, they are just some arbitrary features. If more regular features were used, like multiplication of each single input, this approach would be similar to what is known as *the functional link*, invented by Pao [Pao 89]. The principle of the Distributed Construction algorithm and its corresponding architecture are shown in figure 6.3.

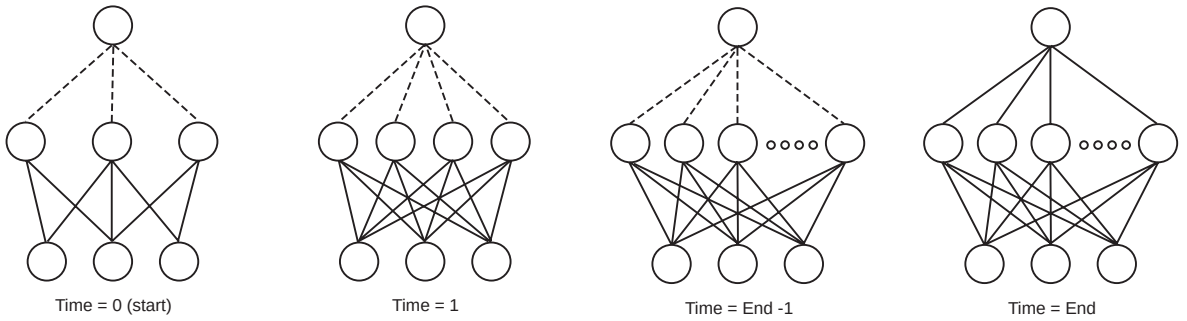


Figure 6.3: Evolution of the Distributed Construction Algorithm. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution.

These three construction algorithms are only described for a single output. For the Tower and Inverted Pyramid a possible extension is to repeat the algorithm for each output, but it is an expensive solution measured in number of connections. Combinations of connections from several previous output units to a current output unit might help, but there is no obvious way to do this. The extension to the Distributed method is simply to add the desired number of output units, because the hidden units are randomly chosen feature detectors. A more detailed description of the Tower and Distributed methods including experiments can be found in [Gallant 90].

The networks are all limited to have units with threshold functions as transfer functions, which makes it possible to use the Pocket Algorithm. However, units with continuous transfer function like the sigmoid can also be applied as long as the training algorithm function also is changed. The properties of convergence will disappear along with the disappearance of the Pocket Algorithm, hence there is no guarantee of convergence, as it is not certain that each unit will make the problem easier for the next output unit.

Another drawback to these methods is that the aspect of generalisation ability versus the number of units needed to get a low error on the training set is not considered.

6.3 Sequential Learning Algorithm

Instead of using hidden units with randomly chosen weights like Gallant did in the Distributed method, it might be a better idea to make sure that the hidden units are actually doing something useful. In [5] they also propose a distributed method where the hidden units are added one by one until the patterns on the hidden layer level are linearly separable. A simple perceptron has proved to be able to solve a linearly separable problem which means that by connecting the output from each unit to an output unit with a threshold transfer function, the problem will be solved. The following description is restricted to the boolean case where there are P patterns with N binary inputs and a single binary output. An easy way to make the problem linearly separable would be to use a single unit for each pattern, so the unit would only be ON when that particular pattern is presented, this is known as a Grandmother cell[3] or match filter. The constructed network with P

[3]The term comes from the idea that the brain might contain cells that fire (give a positive response) only when encountering one's grandmother.

hidden units will guarantee convergence, but its generalisation ability can be expected to be low as a consequence of the high total number of parameters and also because the network has not learned anything, it simply memorises the patterns. The goal is then to find a small number of hidden units which are able to make the P patterns linearly separable. [5] solved this problem by using the following algorithm:

1. **Initial configuration**: The algorithm begins with a single hidden unit (a simple perceptron) with N input units and a single output unit. N is given on the basis of the problem that the network is to learn. Two counters p and i, counting the number of unlearnt patterns and the number of hidden units, are set to P and 1 respectively.

2. **Training**: The hidden unit is trained on the p patterns so that it has the same output (say +1) for all patterns with a target +1 and the opposite output (-1) for p_i of the remaining patterns with the opposite target -1 (p_i will always be at least one).

3. **Reducing the training set**: The p_i patterns with target -1 that were correctly classified are then removed from the training set used by the next hidden units to be inserted and p is set to $p - p_i$. If the remaining p training patterns all have the same target value go to step 5.

4. **Insert a new hidden unit**: Freeze all weights to the current hidden unit, add a new hidden unit to the network and connect it to all input units, and increment i by one. Go to step 2.

5. **Insert the output unit**: Insert an output unit and connect it to all the hidden units.

6. **Train the output unit**: Use the perceptron learning rule to train the output unit and stop.

The principle of the algorithm and its corresponding architecture are illustrated by showing how the algorithm solves a problem, see figure 6.4.

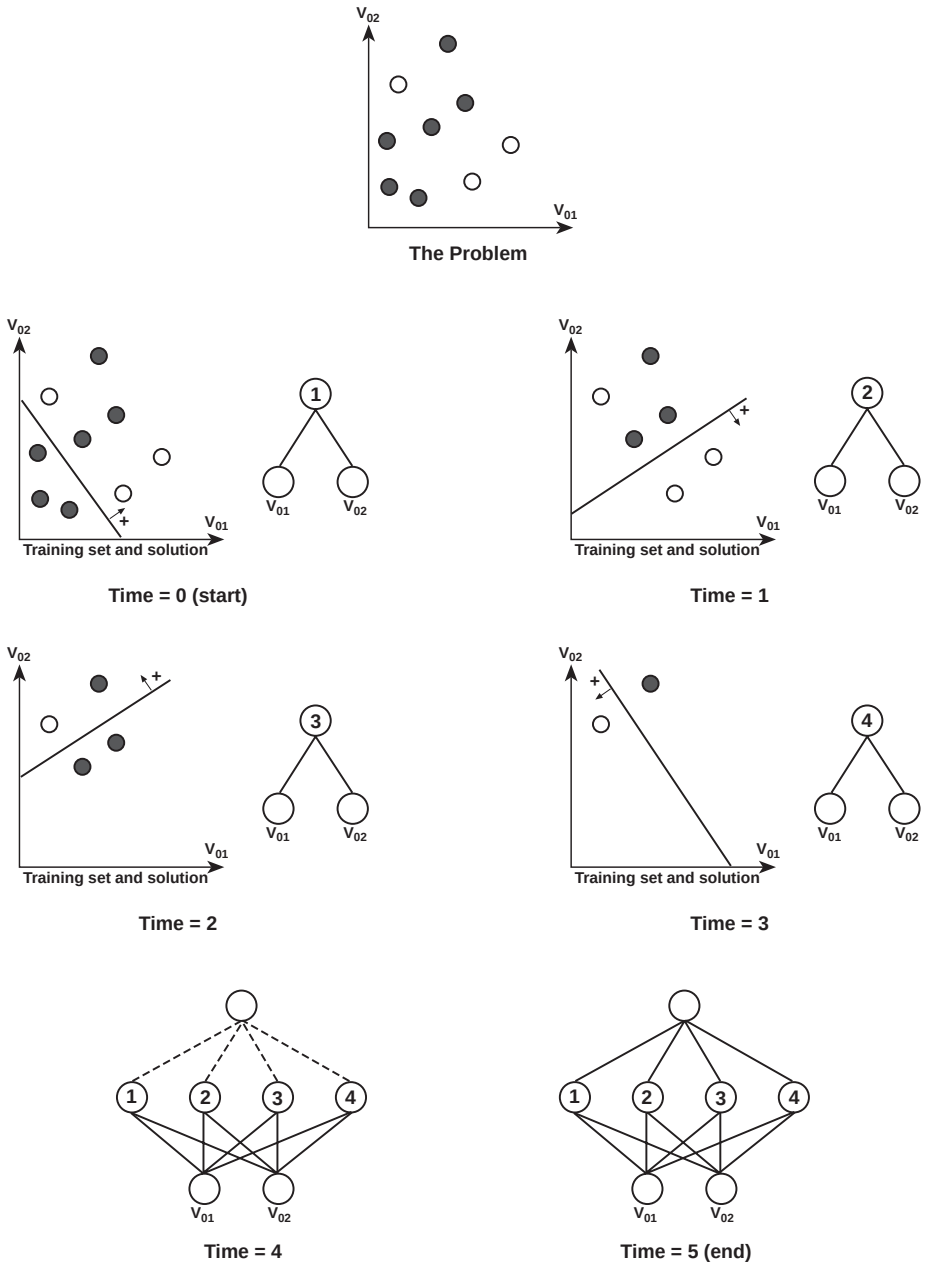


Figure 6.4: Evolution of the Sequential Learning Algorithm and its solution to a problem. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution. White indicates that the pattern belongs to class 1 and black that it belongs to 0.

Two things should be noted here: first convergence is guaranteed if Grandmother cells were used, but this is not interesting because the network is then memorising and not learning anything which makes the generalisation ability poor. The other thing is that the training set is changed for every hidden unit. The weights to the output can be calculated directly. The drawbacks on the other hand are that this method is only valid for the boolean case having both binary input and a single output, and it is not able to handle data sets where misclassifications are present.

6.4 Cascade-Correlation Algorithm

Among some of the more popular methods the Cascade-Correlation Algorithm (CCA) has attracted most attention. The reason might be that the CCA works with continuous transfer functions such as *hyperbolic tangent*, in contrary to many of the others which only work with binary units. Even though convergence has never been proved for the CCA, many reports on experimental results have proved the value of the CCA.

The Cascade-Correlation Algorithm generates a special type of feed-forward network, similar to the Inverse Pyramid architecture, with only one unit in each of the intermediate layers. The main idea is to start by training a simple perceptron to perform as well as possible. If the error is larger than some acceptable error, a new hidden unit is selected from a *pool* of candidate units and inserted. Before a candidate unit is inserted all candidates are first connected to all input units. Then each candidate is trained to maximise the magnitude of the correlations between its output and the residual error signal of the net. The residual error is defined as the sum of all differences between each target and output from each output unit. Once this is done the candidates' weights are frozen. The best performing candidate is selected and its output is finally connected to the output units and the net is retrained, starting from the current "weight position". If the network is still unable to obtain an error lower than the acceptable error, a new pool of candidates, not only connected to the input units but also to the previous candidate is trained. The new candidate that is inserted is thus placed in a layer above the previous candidate. This procedure of inserting and training candidate units is repeated until the error is below the acceptable error. This way of constructing leads to a network in which the hidden units become increasingly higher-order feature detectors. The principle of the Cascade-Correlation learning algorithm and its corresponding architecture are shown in figure 6.5.

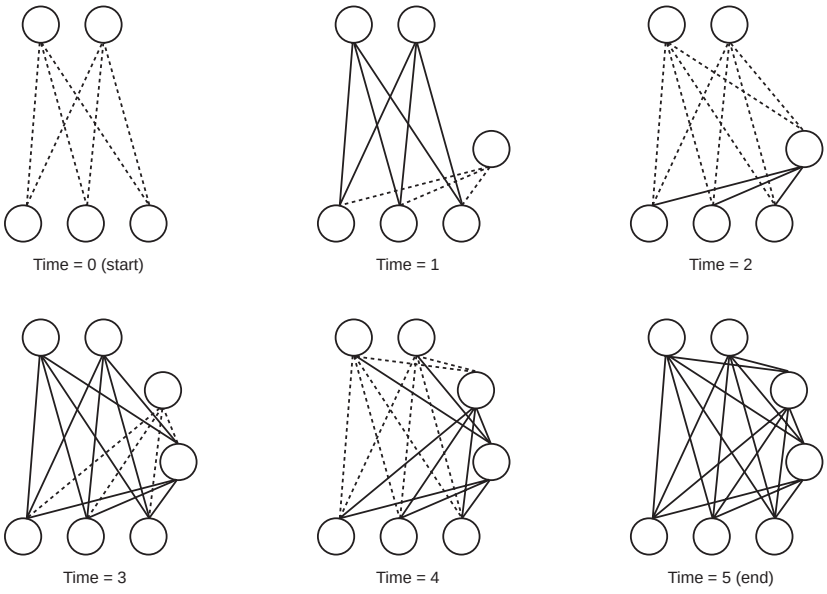


Figure 6.5: Evolution of the Cascade-Correlation Algorithm. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution.

The motivation for inserting a single unit at a time and freezing its weights is as mentioned in paragraph 6.3 that it makes the training of the network very quick. In an ordinary feed-forward network the updating of weights in the lower layers normally takes a long time, because the error that is back-propagated through the network decreases in the lower layers thereby making the updating of the weights small. This also prevents what is known as *The Moving Target Problem*. The term covers the phenomenon which is often observed when training a multi-layer perceptron with the Back-Propagation algorithm. Each unit in the network is trying to extract some features from the input that will improve the network's overall performance. But when several units are trained at the same time, they will compete on extracting features, defined by the back-propagated error. But this error will change for a while, since every node is evolving into a feature detector. Thus until a sufficiently large part of the units decide on what feature to extract, a lot of time is spent on finding a useful feature for each unit, instead of learning that feature.

The Cascade-Correlation Algorithm is as follows:

1. **Initial configuration**: The algorithm begins with a simple perceptron with N input units and M output units. N and M are chosen on the basis of the problem that the network is to learn. The weights are randomly initialised.

2. **Initial training**: The perceptron is trained on the entire training set $\{(\mathbf{V}^p, \mathbf{T}^p) \mid p = 1, \ldots, P\}$, until the performance of the network is as good as possible. If the desired performance is obtained, go to step 6. Otherwise: Start adding hidden units to the network, one by one.

3. **Training of candidates**: A *pool* of candidates for a new hidden unit is generated. This pool emulates a stochastic search in the weight space, which will decrease the risk of inserting a candidate stranded in a local minimum with high error. Each node in the pool of candidates is connected to all input nodes and all previously inserted hidden units. Each candidate is trained with the purpose of maximising the magnitude of the correlations between its output and the residual error signal of the net.

4. **Inserting a new hidden unit**: The candidate with the highest score is inserted "for real" in the network as a new hidden unit. The incoming weights to the new hidden unit are then *frozen*, i.e. they are not to be changed anymore. The new hidden unit is connected to all output nodes with random weights.

5. **Retraining the network**: All incoming weights to the output units are retrained in order to adjust the weights from the newly inserted hidden unit. If the network's performance is satisfying after retraining, go to 6. Otherwise: Go to 3.

6. **Stop**: The algorithm stops.

The CCA has been reported to be very efficient in the task of learning almost any classification problem [Fahlman and Lebiere 90]. The constructed networks' ability to generalise has, however, not shown the same convincing results. The problem is that the CCA often inserts a lot of unnecessary hidden units. The reasons are among other things:

when considering the training of the candidate, the CCA employs the covariance$^{\heartsuit}$ instead of the computational demanding correlation, and this is a bad approximation unless the nature of the problem is such that the candidate should be trained to saturation. It is not at all clear how to train the output units to a point optimal for candidate insertion. One thing is, however, certain: if the output units are trained to saturation the effect of inserting the candidate will not be reduced. A detailed description and analysis of the CCA, including mathematical foundation, description of some major problems and an attempt to solve them, can be found in Appendix E.

6.5 Restricted Coulomb Energy Network

The Restricted Coulomb Energy (RCE) network is the name of the architecture that comes up when the algorithm for automatically building networks proposed in [Reilly *et al.* 82] is used. This method will therefore be called the RCE method. The RCE network is basically what today is called a Radial Basis Function network, see chapter 2. It consists of a single hidden layer of units whose transfer function is a pulse function and an output unit for each class. The RCE method is a simple cluster method based on the idea of clustering the patterns into regions specified by a number of prototypes which are then combined by OR gates. In a two-dimensional case the prototype is an RBF unit with a step transfer function with a predetermined radius and a centre determined by a pattern's placement in the feature space, so the region of the prototype is a circle. The radii of the circles are then reduced until they only contain data belonging to the same class. Each cluster is then constructed by an OR gate that joins together circles from the same class. The RCE algorithm is:

1. **Initial configuration**: The algorithm begins by specifying M output units with threshold transfer functions without any connections and a single hidden RBF unit connected to the N inputs. N and M are given on the basis of the problem that the network is to learn. All patterns are put on a queue. A flag F_{ch} indicating if any change has occurred is set to 0.

2. **Initial training**: The first pattern is presented to the network and removed from the queue. The weights of the RBF unit, representing the centre point, are determined (equal the input) and the output from the RBF unit is connected to the output unit representing the class of the pattern.

3. **Load pattern**: If there are elements in the queue, present the next pattern from the queue to the network and remove it from the queue, else go to step 7.

4. **Evaluate**: If the output of the network is correct go to step 3. If there is no output from the network set F_{ch} to 1 and go to step 5. If the output is wrong set F_{ch} to 1 and go to step 6.

5. **Insert a new hidden unit**: Add a new hidden unit, set the centre point equal to the pattern that gave no response, connect it to the output that represents its class and go to step 3.

6. **Reducing the radius**: The radius of the hidden units that cause the error for that particular pattern is reduced until the error disappears. Go to step 5.

7. **Update queue**: If F_{ch} is different from 0 (there have been some changes) put all patterns on the queue and go to step 3. Otherwise go to step 8.

8. **Stop**: The algorithm stops.

Two things should be noted: first that the value of the radius is never increased, although it would be possible. The reason is that many of the radii may already have been decreased and the risk of making more errors than getting correct responses is considered to be less advantageous than just inserting an additional unit. Secondly it is not necessary to train the weights of the connection to output units, they are simply given a value that will make the output of the output unit ON if just one of the hidden units is also ON, thus making an OR function for the hidden units connected to it.

The principle of the RCE algorithm and its corresponding architecture are illustrated by showing how the algorithm solves a problem, see figure 6.6.

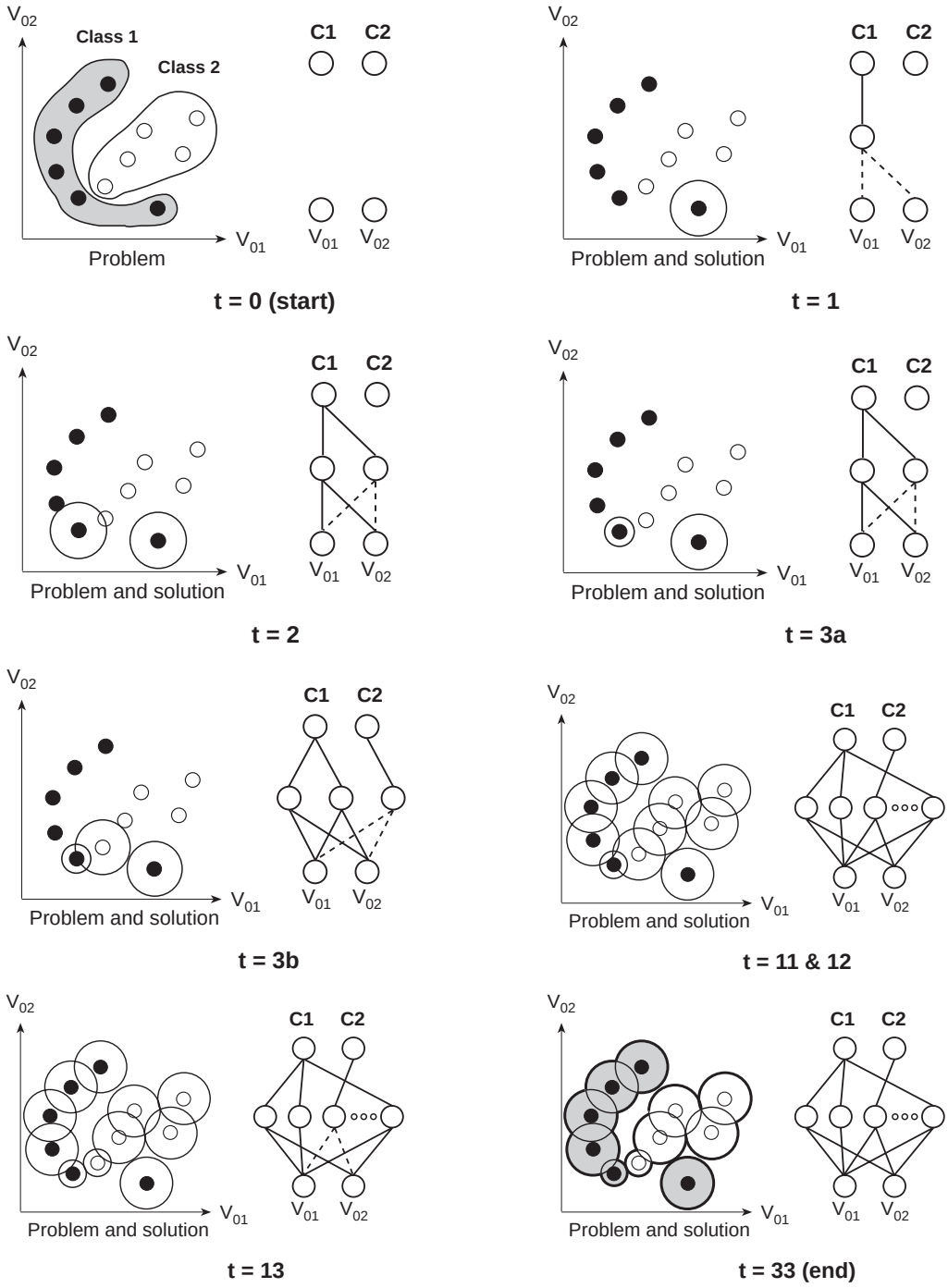


Figure 6.6: Evolution of the RCE Algorithm and its solution to a problem. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution.

6.6 RAN

In 1991 Platt came up with a network that he called Resource-allocation network (RAN) [8]. The RAN network is a two-layer RBF network with short cuts from the input units to the output units. The first layer consists of an unspecified number of RBF units with Gaussian transfer functions[4] so they only respond to local regions of the input space. The number of RBF units will be decided by the accuracy needed, during learning. The output layer consists of linear units, which aggregate the output from the RBF units and create a function that approximates a global input-output mapping. The output from an output unit is given by:

$$V_{Li} = \sum_{j=1}^{N_{RBF}} w_{ij}\Phi(\parallel V_{0k} - c_j \parallel) + \sum_{l=0}^{N_0} w_{il}V_{0l} \tag{6.1}$$

where w_{ij} and w_{il} are the weight from the $j'th$ unit in the hidden layer and the weight from the $l'th$ unit in the input layer to the $i'th$ unit in the output layer L respectively, and N_{RBF} is the number of RBF units at the time in question.

The idea of RAN is first to create a coarse representation of the mapping function to be learned and then refine the representation by allocating RBF units with smaller and smaller widths. The allocation of an RBF unit is done if the error between the output of the network and the target is smaller than a given value ϵ or if the distance between the pattern and the centre of the nearest RBF units is smaller than a given value $\delta(t)$. This two-sided memorisation condition is necessary in order to create a compact network. If the "distance to nearest centre" condition only is used, the network will allocate units instead of using Gradient Descent to correct small errors. If the "total error" condition only is used, fine-scale units may be allocated in order to represent coarse-scale features, which is wasteful.

The value ϵ determines the accuracy of the output of the network. Errors larger than ϵ are immediately corrected by allocation of a new unit, while errors smaller than ϵ are gradually repaired by using Gradient Descent. The distance $\delta(t)$ is the scale of resolution that the network is fitting at the t'th input. The learning starts with $\delta(t) = \delta_{Max}$ which is the largest length scale of interest, typically the size of the entire input space of non-zero probability density. The distance $\delta(t)$ shrinks until it reaches δ_{Min}, which is the smallest length scale of interest. $\delta(t)$ is determined by the following function:

$$\delta(t) = \max\left(\delta_{Max}\exp(-t/\tau), \delta_{Min}\right) \tag{6.2}$$

where τ is a decay constant.

The update of the weights is performed by using the Gradient Descent in two steps, first for the centre of the RBF unit and then for all weights to the output unit.

[4]Platt used the following continuous polynomial approximation of the Gaussian transfer function to speed up the learning: Gaussian output $\Phi = (1 - \sum_k^{N_0}(V_{0k} - c_j)^2)/(q\sigma^2)$, if $\sum_k^{N_0}(V_{0k} - c_j)^2) < q\sigma^2$ otherwise $\Phi = 0$. q is empirically chosen to 2.67 in order to give the best fit.

An algorithm for RAN is as follows:

1. **Initial configuration**: The algorithm begins with a simple perceptron with N input units and M output units. N and M are chosen on the basis of the problem that the network is to learn. The weights are randomly initialised.

2. **Evaluate**: A pattern is presented to the network. If the error between the output of the network and the target is smaller than a given value ϵ or the distance between the i pattern and the centre of the nearest RBF unit is smaller than a given value δ, take a new pattern. If these two conditions are satisfactory for all patterns go to step 5.

3. **Insert a new hidden unit**: Add a new hidden unit, set the centre point equal to the pattern that gave too large a difference and connect it to all output units with random weights.

4. **Retraining the network**: Retrain all the incoming weights to the output units in order to adjust the weights from the newly inserted hidden unit and go to 2.

5. **Stop**: The algorithm stops.

The principle of the RAN algorithm and its corresponding architecture are shown in figure 6.7.

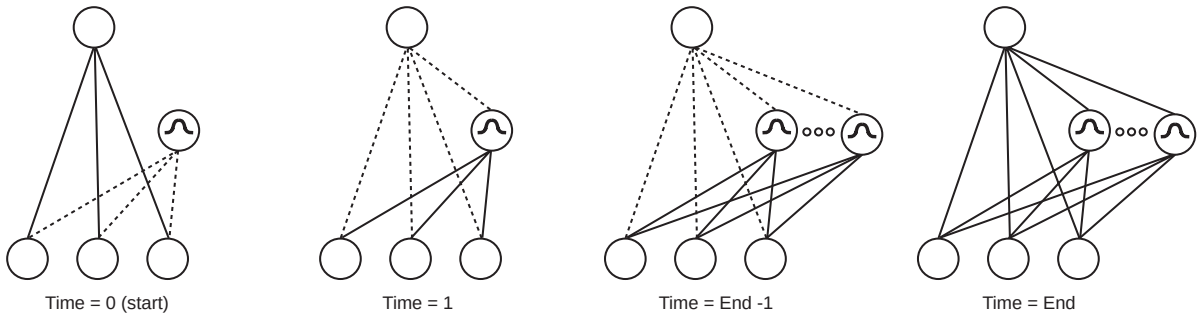


Figure 6.7: Evolution of the RAN Algorithm. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution.

6.7 Global-Local Learning Algorithm

Inspired by the previously described methods Depenau proposed in '95 a new method which applied a Global-Local approach, combining units with sigmoid and Gaussian transfer functions in the hidden layer to solve classification tasks, see [Depenau 95] (Appendix F). The Global-Local approach, or simply GLOCAL, is another supervised learning method for dynamically building and training neural networks. The overall philosophy of GLOCAL is: first do as well as possible for as many patterns as possible in a simple way and then apply special treatment, i.e. use a more refined method for those patterns causing problems.

The GLOCAL method starts with a **glo**bal classification by training a simple one-layer perceptron to perform as well as possible. The training of the perceptron can be performed by Back-Propagation (BP) or another learning algorithm that is able to minimise an appropriate error function. Once the training of the perceptron has ended, the weights are frozen and all patterns that failed to be learnt are noted. These unlearnt patterns are then corrected with a lo**cal** classification made by inserting and training a suitable number of radial basis functions (RBF). The global and local classifications are combined in a new output layer so the final network becomes a Multi-Layer Perceptron (MLP) with one hidden layer that contains both sigmoid and RBF units.

6.7.1 Clustering

The first question that arises is that of the number of RBF units needed. It is trivial to place the centre point of a unit on every pattern and then use a pulse function that only gives a response when that pattern or similar patterns are presented. This is not desirable for many reasons: the errors are memorised instead of being learnt and therefore a low generalisation ability can be expected, and it becomes practically infeasible when the number of errors becomes large. To avoid this problem, a sort of clustering algorithm that will place patterns that are close in the input space and belong to the same output class in the same cluster can be used. There are several different iterative methods that perform the task of clustering$^{\heartsuit}$, see e.g. [Duda and Hart 73](Chapter 6), [Fukunaga 90](chapter 11), [Dasarathy 90] and [Schalkoff 92](chapter 5). Many of these cluster algorithms including the very popular "*K-mean algorithm*$^{\heartsuit}$" are based on the neighbourhood properties among the patterns. The problem with using one of these methods is that in general they do not take the class membership into consideration. Below a new simple cluster algorithm that takes the class membership into consideration is summarised.

1. **Initial configuration**: Assign each training pattern that the perceptron failed to learn as the centre point of a unique cluster and label it (K = 1...C).
2. **Select a cluster.**
3. **Cluster elements**: Find the training patterns that belong to the same class as the cluster, and have a distance that is smaller than the nearest training pattern from another class.
4. **Calculate centre points**: If new elements are clustered, calculate a new centre point. If one of the other cluster centres becomes a member of the current cluster, remove it from the list of clusters. Return to step 3.
5. **Stop**: If there are untouched clusters on the list, return to step 2, otherwise stop.

6.7.2 Combining RBFs

Once the centre position is placed, the RBF unit's parameter is determined. How this is done depends among other things on the choice of Φ. It may be calculated or trained, but the general idea is to adjust some radii Σ_{Ndim} in order to control the overlap between different clusters. $\Sigma_{N\times N-dim}$ is the covariance matrix that determines the shape

of the cluster, see e.g. [Duda and Hart 73](pp.22-32). The most simple way to calculate the $\Sigma_{N\times N-dim}$ is to set the diagonal elements of $\Sigma_{N\times N-dim}$ ($\sqrt{\sigma_{ii}}$) equal to the distance between the centre of the cluster and the farthest member and zero otherwise. The RBF unit will thus represent a hyper-sphere in the N-dimensional input space. If Φ has a step transfer function, for patterns inside the hyper-sphere the output will be 1, outside it will be 0. If Φ has instead a Gaussian transfer function, the response will be a gradually decreasing function that depends on the radii. [6] have shown a way to approximate the radii of hyper-ellipsoids, but in the following description the simple method is used. Having determined the RBF units, the outputs from all units are now connected to units in a new output layer. The number of units in the new output layer is of course equal to the number of classes. The activity function of the new output units can be sigmoid or linear. The final network will then become an MLP, with both sigmoid and RBF units in the hidden layer. The principle of the GLOCAL learning algorithm and the evolution of its corresponding architecture are shown in figure 6.8.

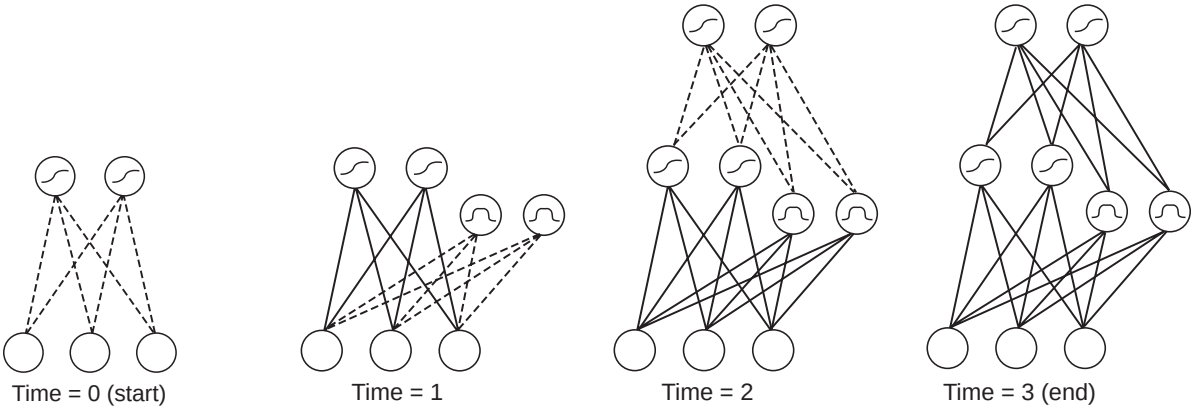


Figure 6.8: An example of how GLOCAL evolves. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution. (t=0) Simple perceptron. (t=1) RBF units are inserted and trained. (t=2) New output units are inserted and connected to all units in the hidden layer. (t=3) Final architecture.

The activity in and the output from one of the output units in the final network become:

$$\text{Activity: } U_{Li}^p = \sum_{j=0}^{M} w_{ij} g(U_{lj}^p) + \sum_{j=M+1}^{N_{RBF}} w_{ij} \Phi(\parallel \mathbf{V}_{0j}^p - \mathbf{C}_j^p \parallel) \tag{6.3}$$

$$\text{Output: } V_{Li}^p = g(U_{Li}^p) \tag{6.4}$$

where w_{ij} is the weight from the $j'th$ unit in the hidden layer to the $i'th$ unit in the output layer L, and N_{RBF} is the number of RBF units.

The pseudo-algorithm for GLOCAL is as follows:

1. **Initial configuration**: The algorithm begins with a simple perceptron with N input units and M output units. N and M are chosen on the basis of the problem that the network is to learn. The weights are randomly initialised.

2. **Initial training**: The perceptron is trained on the entire training set $\{(\mathbf{V}^p, \mathbf{T}^p) \mid p = 1, \ldots, P\}$, until the performance of the network is as good as possible. If the desired performance is obtained, go to step 6. Otherwise: Freeze the incoming weights to the perceptron and mark the patterns that were not learnt.

3. **Cluster and insert hidden RBF units**: Start the clustering algorithm. Add a new hidden RBF unit for each cluster and set the centre point of the RBF unit equal to the centre of the cluster. The radii are all set to the distance between the centre of the cluster and the farthest member.

4. **Insert output units**: Insert M output units and connect them to all units in the hidden layer and initialise the weights.

5. **Training the network**: All the incoming weights to the output units are trained. The radii and sometimes also the centre of the RBF units may also be trained depending on the RBF units' transfer functions.

6. **Stop**: The algorithm stops.

If Φ is chosen so it will only give a strong response to patterns inside its boundaries or zero elsewhere, the use of the cluster algorithm described earlier will result in a number of RBF units that are able to correct the errors 100% and not damage any of the previous correct classifications made by the perceptron. This means that the representation at this level is linearly separable. Since the absolute value of $g(U_{lj}^p)$ can never exceed 1, a **correct output is guaranteed** by only connecting the perceptron unit and the RBF units that represent the same class to the new output unit representing this class and then setting the weight between each RBF unit and the output unit class w_{ij}^{Phi} to a positive value larger than the sum of all weights between the original perceptron units and the output unit, i.e. $w_{ij}^{Phi} - \sum_{j=0}^{M} |w_{ij}| > 0$. This means that GLOCAL will converge to zero error while learning any consistent classification of real-valued input. If Φ is chosen with a Gaussian transfer function, there might be an overlap between different clusters, so patterns belonging to other classes may also cause some smaller response from RBF units of opposite classes. The effect of this overlap is controlled mainly through adjustment (training) of the radii, but also through training of the output connection. The training can be performed by Back-Propagation until the error measurement is as low as possible, but there is no longer any guarantee that GLOCAL will converge to zero error.

Experiments with GLOCAL using RBF units with Gaussian transfer function showed that it is vital for the performance that the weights of the connections to the output units are initialised in a special way. The reason is that the update of the weights depends on the output of the hidden units. During an online-learning the first perceptron units will be active all the time while the RBF units will be zero for most patterns. The weights between the perceptron units and the new output units will be updated all the time while the weights between RBF units and the new output units rarely is updated. If the weights are initialised randomly to small values, there is a tendency that the weights between the perceptron units and the output units will decrease in magnitude and totally control the response from the output units, while the weights between the RBF units and the output units stay almost the same. A way to compensate for this is to initialise the weight

between the RBF units and the output unit representing the class to a large positive value. The size of the value depends on the problem and how many patterns that make the output of the RBF units high[5].

Besides the convergence property the additional advantage of choosing Φ with a pulse transfer function is that there is no need for extra output units. The output response from the RBF units can be connected directly to the output units of the perceptron. If the weight condition "$w_{ij}^{\Phi} - \sum_{j=0}^{N_0} w_{ij} > 0$" is fulfilled, the RBF unit will always control the total output for the patterns that the perceptron was unable to classify correctly. The principle of the GLOCAL learning algorithm and the evolution of its corresponding architecture will then be changed as shown in figure 6.9.

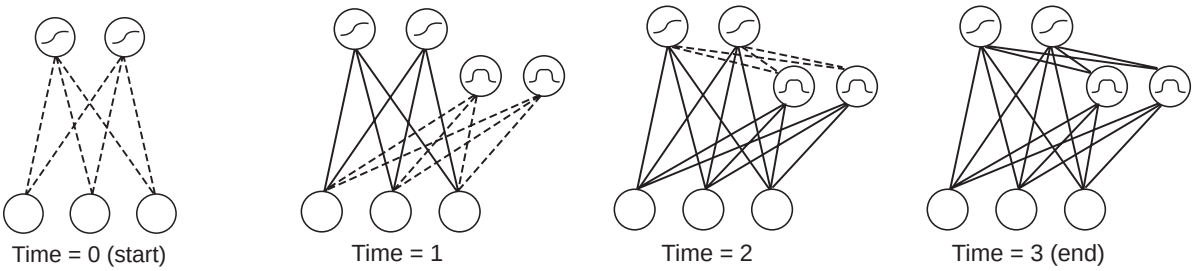


Figure 6.9: An example of how GLOCAL using Φ with a pulse transfer function evolves. Dotted (solid) lines indicate that the weights are changed (not changed) in this phase of the evolution. (t=0) Simple perceptron. (t=1) RBF units are inserted and trained. (t=2) Connect and train the weights between the output and hidden layer units. (t=3) Final architecture of an alternative solution when short cut is allowed, so the final network becomes a cascade network.

Although the above choice of Φ and straight forward calculation of the weights mean that GLOCAL will converge to zero error while learning any consistent classification of real-valued input, it is not necessarily a good idea. The problems come when the data set contains contradictory patterns i.e. patterns with same input but different target. One way of handling this problem is simply to use RBF units with Gaussian transfer functions and allow training of the weights, so the centre positions in the input space are allowed to move. The proposed cluster algorithm will cause the insertion of units located in the same place in the input space. By training the centre position to give the smallest error the centre will be placed where it is optimal.

Another way of handling contradictory patterns is to let the cluster algorithm take care of it. It will only take a minor modification of the cluster algorithm to make it abstain from inserting an RBF unit if the distance between two input patterns with different targets is zero. This simple idea gave inspiration to a number of modifications of the cluster algorithm. For example, if the distance between two input patterns with different targets was smaller than some value, no RBF units should be inserted. A further development of this idea lead to a strategy for reducing the number of RBF units in order to improve the generalisation ability. The strategy was to allow a certain percentage

[5]A weight whose value is at least the sum of the magnitude of all weights to the output unit seems to work well.

of the cluster elements to belong to another output class. This idea was explored and described in [Pedersen 95] who found through a number of experiments that the number of inserted RBF units was reduced, but the generalisation ability (test set error) was not improved. The final example was the idea of only inserting an RBF unit if the cluster contained more than 2 elements. This can be interpreted as follows: if a cluster contains only one element, the pattern is probably wrongly classified in the training set.

6.7.3 Implementation and Experiment

In order to illustrate how GLOCAL works, a non-linearly separable two-dimensional classification problem with 3 classes was constructed. The problem and the GLOCAL solution are shown in figure 6.10.

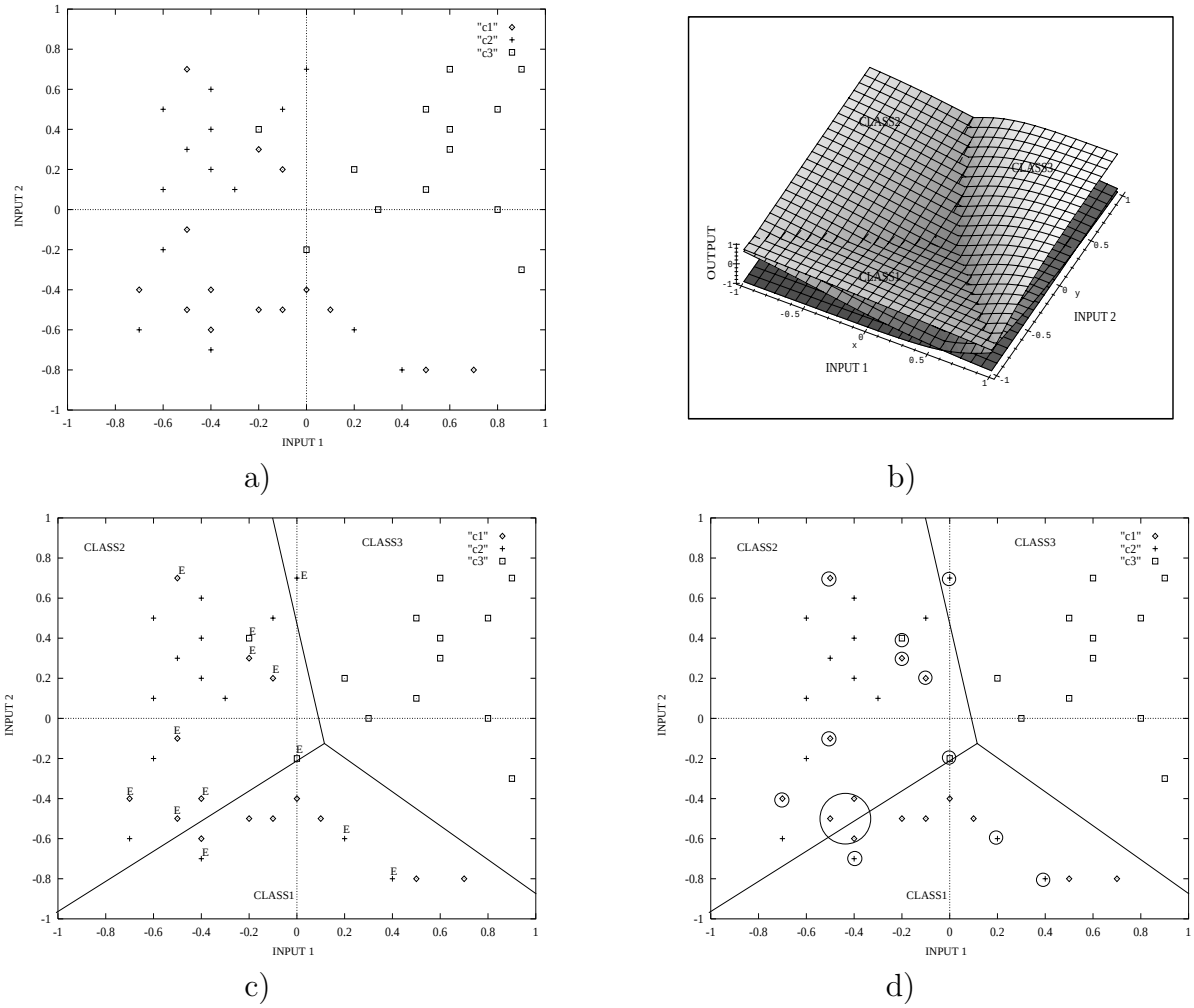


Figure 6.10: a) The three-class classification problem. b) The result of training the first perceptron (in 3-dim). c) The result of training the first perceptron including the errors projected to 2-dim. d) The final solution.

In order to show the potential of GLOCAL, a simple version was tested on a Benchmark

problem, the cancer1 problem[6], taken from the PROBEN-1 database [9]. The reason for choosing this problem is that it is real life data and that the input dimension has a size that is high enough to show "high-dimensional" phenomena. The task is to classify a tumour as either benign or malign based on cell descriptions gathered by microscopic examination. The GLOCAL method was implemented in the following way: the initial perceptron was trained by a standard BP with a learning rate $\eta = 0.01$ and a momentum term $\alpha = 0.8$. Φ was a pulse function with a uniform radius in the Euclidian norm that was set equal to the distance between the centre of a cluster and the farthest pattern within the cluster. The output units were also trained by BP with $\eta = 0.001$ and $\alpha = 0.9$. For the sake of comparison the CCA was tested as well. The original code made by Fahlman [Fahlman and Lebiere 90] was used with the following parameters: $\eta = 0.1$, *offset* $= 0.1$, the *"patience factor"* $= 0$ and *poolsize* $= 1$. The training set contained 525 patterns and the test set 174 patterns. No validation set was used. Both CCA and GLOCAL were trained until the training set was learned. A pattern was considered to be correctly classified if it diverged no more than 0.8 for the training set and 0.9 for the test set. The initial network had 9 input units and 2 output units. The CCA used between 1004 and 2435 epochs to learn the task while the GLOCAL was preconditioned to always run 150 epochs for the perceptron and the output layer. 60 trials were run for each experiment. The result along with the result for the MLP taken from [9] are shown in table 6.1.

	Architecture			Number of Weights		Training Set (Classification Error in %)		Test set (Classification Error in %)	
	max	mean	min	max	min	mean	stddev	mean	stddev
MLP	-	4+2	-	48	48	2.83	0.15	1.38	0.49
CCA	14x1	10x1	6x1	263	99	0	0	4.3	1.3
GLOCAL	-	20	-	220	220	0	0	1.72	0

Table 6.1: Performance table for an MLP, the CCA and GLOCAL on the cancer1 data. *X+Y means X units in the first hidden layer and Y in the second hidden layer, while* $X \times 1$ *means X hidden layers with one unit.*

From table 6.1 it can be seen that GLOCAL performs much better than the CCA and almost as well as the MLP. This is remarkable because the architecture of the MLP was chosen as the best among 12 different network topologies and was further optimised by using a validation set in the training. A version of GLOCAL where Φ was Gaussian is able to bring the test error down to 1.2% in single cases, but both the mean and standard deviation (stddev) increased.

Several similar tests on other real life problems taken from the PROBEN1 database showing the performance of both GLOCAL and a version of GLOCAL where the cluster was allowed to contain a percentage of members from another class can be found in [Pedersen 95]. The results from these tests were that the performance of GLOCAL was on average slightly better than an MLP, chosen as described above, and the two GLOCAL methods had the same performance but only the one with the cluster algorithm allowing members from another class was able to handle misclassifications (same input - different output) in the training set. In the next chapter the performance of GLOCAL is shown in connection with classification of signals in radar pictures.

[6]Many thanks to Dr. W. H. Wolberg, University of Wisconsin Hospitals, Madison, for making the cancer data available.

6.8 Résumé, Open Questions and Remarks

This chapter has presented several supervised construction methods/algorithms with units in the hidden layer having either threshold function/sigmoid function or Radial Basis Function as transfer functions. They all served as inspiration to a new construction algorithm that combines units with both types of transfer functions in the hidden layer. This algorithm called *the GLOCAL Algorithm* was described in detail.

The temporary experience with GLOCAL is encouraging, it has an easy analytic traceability, due to its modular construction and clustering nature, and its performance in experiments and tests is comparable to the performance of networks constructed on a more classical approach i.e. where the design is basically done on qualified and academic guess and some rules of thumb that are repeated until success. The price to pay for having GLOCAL automatically building a network is, like for all other construction methods/algorithms, that the number of learning/method parameters that have to be determined in advance is increased. Parameters like number of RBF units allowed to be inserted, number of elements in a cluster before an RBF unit is inserted, parameter to control overlap, etc., all of great influence on the performance of the final network, have to be predetermined.

GLOCAL is currently under further development. Future work will include new, faster and better cluster strategies and algorithms and an analysis of how the number of RBF units can be controlled when the perceptron is trained with Early Stopping.

One of the questions left open with local methods is which metric should be used to measure the distance between points in the input space. Most methods that are based on some local measurement, including the ones in this chapter, assume that the metric is the Euclidian distance (2-norm). This is not necessarily the best. In the case of supervised clustering the optimal solution would be to find a metric, probably a nonlinear weighted distance measure, so that two points from different classes having a distance between them near zero in the Euclidian norm, would have infinite distance in the new metric. This would open the way to the use of a large number of well known cluster methods. Another question left open is how to deal with higher dimensionality in the input space, which also has influence on the metric to be used. Discussions and illustrations of these problems can be found in [Duda and Hart 73], [Schalkoff 92], [4], [6] and [Ripley 94]. It should be stressed that the answers are not trivial, but indeed very important for the overall performance, perhaps the most important.

An alternative approach to building neural network is to use Genetic Algorithms, as suggested in [Miller *et al.* 89], [Harp *et al.* 90] and [Whitley *et al.* 90]. GAs have shown promising results, but similar to the case where they were used for training the network, see paragraph 3.2.7, their practicability is limited to smaller problems. Recent work carried out at the Computer Science Department of Aarhus University by [Arentoft 95] has confirmed this tendency.

Chapter 7

Classification of Ice

The estimation of ice type concentrations from Synthetic Aperture Radar (SAR) images has been investigated for several years, see e.g. [Skriver 94]. The classification estimation has been performed by training a Bayesian Maximum Likelihood Classifier ($BMLC$) [Niblack 85] with a classification rate about 80%. This chapter describes the work done in connection with a preliminary investigation of a neural network's capability to classify ice types. It includes a short review of earlier used techniques, an analysis of data, implementation of different neural networks and results from various experiments with these networks. The neural networks considered are all of the feed-forward type. For training, different learning algorithms and error functions are used. Both pruning and construction algorithms are used to get an optimal architecture. Experiments showed that almost any kind of neural network, using a Standard Back-Propagation (Std_BP) learning algorithm for minimising the Mean Square Error (MSE), is able to perform better than the $BMLC$. The reason is that the neural network is able to use a larger training set than the $BMLC$.

7.1 Introduction

Microwave remote sensing is an essential tool for studying large, medium, and small-scale sea ice phenomena in the polar regions. This is due to the fact that microwave remote sensing is largely independent of light and weather conditions. Synthetic Aperture Radar (SAR) has fine spatial resolution that enables detailed imaging of the scene of observation, and data from ERS-1 have already shown great potential for sea ice mapping. A number of features, which is of great importance for research and application purposes, may be derived from the radar data. This applies to ice concentration, ice motion, lead structure, flow size distribution, etc. [Skriver 94].

The basis of feature retrieval of any kind is the ability to discriminate between the targets in question. In SAR images the basis of discrimination is the properties of the backscatter coefficient, i.e. the mean backscatter coefficient and the spatial variations in the backscatter coefficient, also called texture. Discrimination based on these properties is disturbed by the speckle noise inherent in SAR images. It is normally necessary to reduce the influence of speckle on the image, thereby reducing the variance of the estimates of the features.

A further analysis of a SAR image or any image can either be done by investigating the whole image by looking at single pixels or by looking at larger areas of interest. The latter is used and a fine segmentation algorithm has been developed for that purpose, see [Skriver 94].

In this chapter the techniques and results from previous work done by Skriver using the classical Bayesian Maximum Likelihood Classifier method are shortly reviewed ([Skriver 94], [Skriver 89]). After that results from using a neural network on the same problem are given.

7.2 What is the Problem?

The European satellite ERS-1 orbits around the earth with an interval of 3 days. The SAR on board ERS-1 acquires images of 100 km x 100 km size with a spatial resolution of 25 m x 25 m. The pixel spacing in the PRI images used is 12,5 m x 12,5 m resulting in approximately 8,000 x 8,000 pixels. The location of six images from different locations in the Greenland Sea is shown on the map in figure 7.1. The data considered in the rest of this chapter originate from the image located in position B.

Figure 7.1: Map showing the location of the ERS-1 images

The image that is produced from the satellite's signal has a high spatial resolution, containing about 8,000 x 8,000 pixels.

The purpose of this application is to decide the ice type by classifying it into 6 classes:

Class 1. **Multi-year ice***(MY)*

Class 2. **New ice type a***(NIa)*

Class 3. **New ice type b***(NIb)*

Class 4. **New ice type c***(NIc)*

Class 5. **Open water***(WA)*

Class 6. **Ice mixture***(MIX)*

Each ice type is an indication of how old the ice is.

7.2.1 Segmentation and Features

At Electromagnetics Institute (EMI), Technical University of Denmark (DTU) a program which is able to segment the images has been developed [Skriver 89]. The segmentation algorithm can be briefly described as: 1) Edge detection, 2) Centre point determination, 3) Segment border determination, and 4) Segment merging.

For each of these areas (segments) 16 features are calculated. These features are listed below. The definition and further explanation of these features can be found both in [Haralick 79] and [Skriver 94].

1. Mean value of the backscatter coefficient in dB
2. Mean intensity, e.g. the pixel value is squared before the averaging
3. Standard deviation of intensity
4. Maximum intensity
5. Minimum intensity
6. Power-to-Mean Ratio (PMR), Normalised second order moment $= \frac{E(X^2)}{E(X)^2}$, where X is the intensity.
7. Normalised 3rd order moment $= \frac{E(X^3)}{E(X)^3}$, where X is the intensity.
8. Normalised 4th order moment $= \frac{E(X^4)}{E(X)^4}$, where X is the intensity.
9. Skewness
10. Kurtosis
11. Angular Second Moment

12. Contrast (CON)

13. Inverse Difference Moment

14. Entropy

15. Sum Entropy (SENT)

16. Difference Entropy

These 16 features represent a possible input to the classifier while the 6 classes represent the output.

7.3 A Traditional Method for Classification of Ice

The classification approach that has been used at EMI is the classical *Bayesian Maximum Likelihood Classification* as defined in the following way ([Niblack 85]):

$$g(c|\mathbf{v}) = \frac{g(\mathbf{v}|c)g(c)}{g(\mathbf{v})} \tag{7.1}$$

where $g(c|\mathbf{v})$ is the probability that a segment with the feature vector $\mathbf{v}$ is in class c, $g(\mathbf{v}|c)$ is the probability that class c has the feature vector $\mathbf{v}$, g(c) is the probability that c belongs to class c, and $g(\mathbf{v})$ is the sum of $g(\mathbf{v}|c)$ over all c. $g(\mathbf{v}|c)$ is commonly considered to be a Gaussian distribution function defined as:

$$g(v|c) = \frac{1}{(2\pi)^{\frac{n}{2}} \|\Sigma_c\|^{\frac{1}{2}}} \exp^{-\frac{1}{2}(v-m_c)^T \Sigma_c^{-1}(v-m_c)} \tag{7.2}$$

where the $\mathbf{m}_c$ and Σ_c^{-1} are mean vector and inverse covariance matrix for each class c estimated from the data in the training set.

Assuming that $g(c)$ is a uniformly distributed function or it is known for each class and using the fact that $g(\mathbf{v})$ is simply a normalisation factor, $g(c|\mathbf{v})$ will become:

$$g(c|\mathbf{v}) = g(c)\frac{1}{(2\pi)^{\frac{n}{2}} \|\Sigma_c\|^{\frac{1}{2}}} \exp^{-\frac{1}{2}(v-m_c)^T \Sigma_c^{-1}(v-m_c)} \tag{7.3}$$

and the class of a segment can simply be classified as belonging to the class that gives the largest value when the feature vector $\mathbf{v}$ is inserted into the equation (7.3).

7.3.1 Construction of Data Sets

Using the segmentation algorithm described in the previous paragraph on the image, the result is an image that is split into more than 20,000 segments. Based on the experience of an expert and general knowledge of the weather conditions and the potential ice types, about half of these segments are classified by visual inspection. Among these segments there are **5,780** which belong to one of the 6 classes of interest distributed in the following way: **3,447** MY**, 108** NIa**, 693** NIb**, 643** NIc**, 397** WA **and 492** MIX**.** Although the number of segments represents only 25% of all segments, the size of the segments constitutes about 50% of all pixels in the image.

The segment's feature is strongly disturbed by noise because the backscatter coefficient inherits the so-called speckle noise in the SAR image. A statistical analysis of the features made by Skriver in [Skriver 89] showed that:

1. The distribution of the backscatter coefficient was a gamma distribution and the difference between the 5% and 95% fractiles was as large as 17 dB (about 50 times difference), when individual pixels were considered.

2. There was a strong correlation among the features of the classes when small areas were considered.

3. The estimate of the PMR feature was only asymptomatically unbiased if only large segments were used to calculate the mean PMR.

Since the Bayesian Maximum Likelihood Classifier only works properly with uncorrelated parameters and the PMR feature is considered to be vital for the classification, it is necessary to reduce the variance of the estimated features and the correlations between the features. This is the same as reducing the influence of speckle in the image, which is normally done by simply using as large areas in the image as possible. In [Skriver 94] the PMR is only included in the classification for segments with more than 1000 pixels, because the PMR is not unbiased for smaller segments, as mentioned above. For small segments only the backscatter coefficient, which is unbiased, is used in the classification.

The above-mentioned large number of manually classified segments is in practical applications unrealistic, because it is a very time-consuming and difficult task. Therefore only 30 of the largest segments were used for the training set in [Skriver 94]. This is of course unfortunate because the calculation of the model's parameters only is based on larger segments (more than 1000 pixels), but it is to be used for all segments. Contrary to custom these segments were not removed from the test set, but due to the small ratio between the training set and test set this is not considered to be a serious problem. This means that the segments or data were split into a **training set with only 30 data** and a **test set with 5,780 data**. The training set consists of 5 elements from each class.

7.3.2 Results from Experiments

Different versions of the model, number of features, have been calculated and tested. It turned out that only a relatively small number of features was needed to get the best result from this model. The features were all calculated so the training set was learned.

The results on the test set with different combinations of the Backscatter coefficient (BS), PMR, CON and SENT, are shown in table 7.1.

Ice Type	BS	PMR	CON	SENT	BS & PMR	BS & CON	BS & SENT
MY	88.7	76.7	33.6	37.9	89.7	71.3	71.3
NIa	77.0	44.1	28.4	20.5	94.2	79.9	78.6
NIb	8.9	27.4	7.9	5.0	5.0	8.7	8.5
NIc	97.3	21.1	12.6	26.9	74.8	78.6	67.3
WA	99.4	88.6	68.8	48.9	96.6	99.5	99.4

Table 7.1: Performance table with different combinations of features. *Note that the ice mixture is not included.*

Combinations that included some of the other 16 features were also constructed, but it turned out that the combination of only two features, BS & PMR gave the best classification[1]. For the two combinations BS and $BS\&PMR$ further results are shown in table 7.2:

	BS (in segments)	BS (in area)	BS & PMR (in segments)	BS & PMR (in area)
Error %	43.7	38.1	23.9	27.1

Table 7.2: Performance table for different combinations of features measured in percentage of error on the test set. The ice mixture class is included.

The classification is often used in a context where it is the percentage of the areas that is interesting, because it is more important to classify large areas correctly than small areas. However, in the following investigations the focus will be on the classification ability and therefore the classification will be reported in segments.

The confusion matrix$^{\heartsuit}$ for the "BS & PMR" classifier is shown in table 7.3:

Ice Type	MY	NIa	NIb	NIc	WA	MIX	Total
MY	3145	220	21	6	1	54	3447
NIa	23	69	15	0	0	1	108
NIb	18	156	295	222	2	0	693
NIc	0	0	5	638	0	0	643
WA	178	0	0	0	218	1	397
MIX	434	21	0	0	4	33	492

Table 7.3: Confusion matrix for BC & PMR. Each row indicates how the elements from the class at the left are classified.

[1] An alternative way to reduce the number of features is to use the Karhunen-Loeve transformation also known as Principal Component Analysis, see e.g. [Schalkoff 92](pp. 306-308)

7.4 Neural Networks for Classification of Ice

7.4.1 Neural Network Architecture

In order to explore neural networks' capability to classify ice types, several different networks were implemented. Four types of feed-forward network were considered, a simple perceptron, Multi-layer perceptron, Cascade-Correlation Architecture (CCA) and the Global-Local Architecture (GLOCAL), see figure 7.2. A detailed description of a simple perceptron and of the Multi-layer network is given in chapter 2, while descriptions of CCA and GLOCAL can be found in chapter 6.

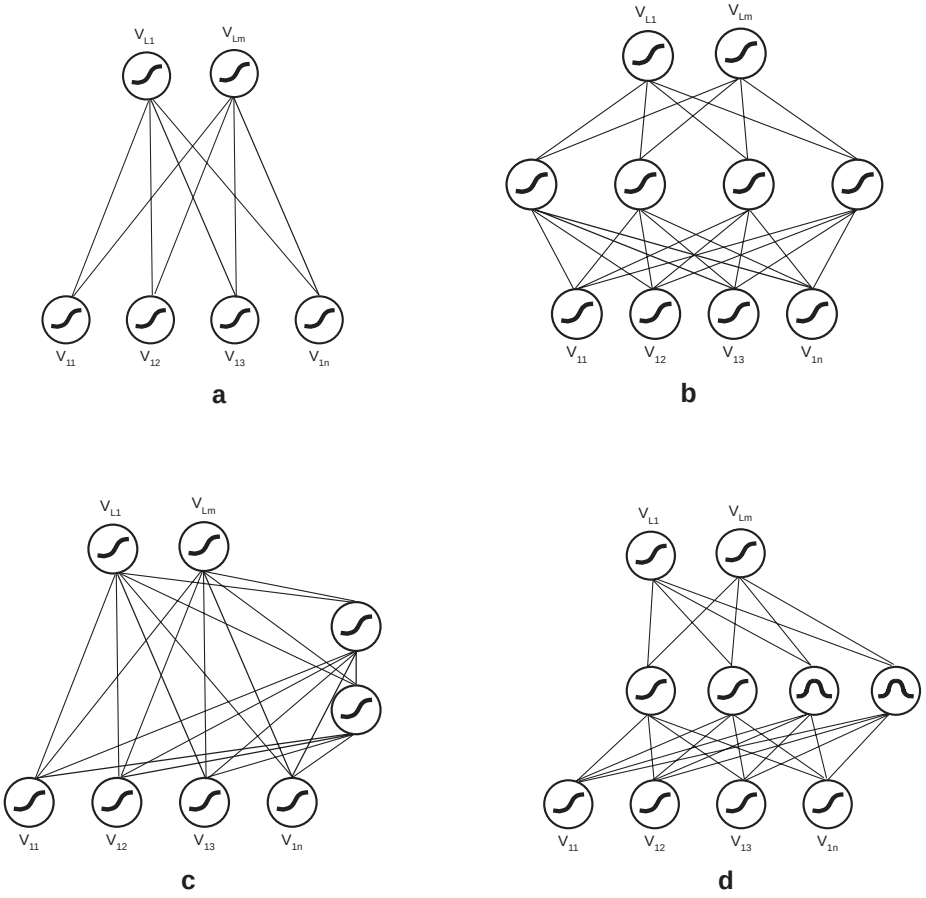


Figure 7.2: Implemented types of network: a) Simple perceptron, b) Multi-layer perceptron, c) Cascade-Correlation, and d) GLOCAL

The implementation of the first three types of network was done in the very popular SNNS simulator [SNNS 94], probably one of the best simulators at the moment[2].

[2]In 1993 Lutzy *et al.* [Lutzy *et al.* 93] made an evaluation of some of the most popular simulators, PlaNet v5.6, Pygmalion v2.0, Rochesster Connectionist Simulator v4.2. In their conclusions they recommend the use of the SNNS.

7.4.2 Preprocessing of Data

Before the data were presented to the networks, an additional preprocessing was made. The data were normalised so both input $\mathbf{V_{In}} = (V_{01}, V_{02}, ..., V_{0N})$ and output $\mathbf{V_{Out}} = (V_{L1}, V_{L2}, ..., V_{LM})$ lay in the interval between V_{MAX} and V_{MIN}, where V_{MAX} and V_{MIN} are -0.9 and 0.9 or -1.0 and 1.0 respectively.

The normalisation was made in three different ways:

1 A linear transformation where each input element V_{1i} is normalised to V_{1i}^{Nor} in the following way:

$$V_{0i}^{Nor} = V_{MAX} * (\frac{V_{0i} - V_{0i}^{Min}}{V_{0i}^{Max} - V_{0i}^{Min}}) - V_{MIN} \tag{7.4}$$

where V_{0i}^{Max} and V_{0i}^{Min} are the maximum and minimum value of the input V_{0i}.

2 A non-linear transformation where the V_{0i}^{Nor} from 1. is centred and passed through a tangent hyperbolic function as follows:

$$V_{1i}^{NorH} = TanH(V_{1i}^{Nor} - \overline{V_{1i}^{Nor}}) \tag{7.5}$$

where $\overline{V_{1i}^{Nor}}$ is the mean value of inputs V_{1i}^{Nor} taken over all training patterns.

3 The V_{1i}^{Nor} from 1 is used as input but the input units are chosen to be the Tanh-type with an offset, and the offset is adjusted during the training of the network.

The normalisation was done to ensure that the input elements are equally weighted and that the dynamism of the hidden units in the first layer is explored. Further description and alternative ideas of normalisation can be found in [Le Cun 93a]. Although this seems reasonable from a neural network point of view, one should be aware that it might make the task even harder. The problem is that points (parameter) that were originally easy to separate because the distance between them was large, may now be placed close to each other. A nice illustration of this problem can be found in [Duda and Hart 73](pp. 213-217). An analysis of these normalised data is given later.

7.4.3 Experiments and Results on the 30/5780 Data Set

In the first series of experiments with neural networks, the ratio between training and test set was kept at 30/5780[3] in order to be able to compare the results directly with the results from the $BMLC$.

A number of different network architectures have been implemented and trained with different learning algorithms and parameters. A representative extract of the networks is shown in table 7.4.

[3] 30/5780 means 30 patterns in the training set and 5780 patterns in the test set

TYPE	Architecture	Number of Weights	Learning Algorithms	Training Set Classification Error in %	Test set Classification Error in %
PERCEP	0	96	Std_BP	0	37.0
PERCEP	0	48	Pru_Mag	0	36.7
MLP	11	242	Std_BP	0	28.7
MLP	7	45	Pru_Mag	0	29.7
MLP	16+3	322	Std_BP	0	32.8
MLP	6+2	56	Pru_Mag	0	36.2
CCA	0	96	Sdt_BP	0	37.0
GLOCAL	0	96	Sdt_BP	0	37.0

Table 7.4: Performance table for an MLP, the CCA and GLOCAL on the SAR data. *X+Y means X unit in the first hidden layer and Y in the second hidden layer.*

General comments on the implementations:

The *BMLC* was able to learn the training set. The requirement to all neural networks, in order to make the result comparable, was therefore that the training should at least continue until the **classification error on the training set was 0**. All implemented networks were able to meet this requirement, which means that the problem given this training set is linearly separable. The three pruning algorithms described in chapter 5 were applied, the **Magnitude Pruning algorithm (MAG)**, **Optimal Brain Damage (OBD)** and **Optimal Brain Surgeon (OBS)**, and the requirement to all of them was that they should stop pruning as soon as the network was no longer able to learn the training set, and then use the weight combination just before the error appeared.

Comments on the implementations:

A **simple perceptron** with 16 inputs and 6 outputs, i.e. 96 weights plus 6 thresholds was designed. The training was done by a standard Back-Propagation and continued until the MSE did not show any progress. **The classification error on the test set was as high as 37.0%**. In order to make the model more general, the number of superfluous weights was reduced.

All pruning methods showed comparable performance, but when the MAG was used the test error was about 0.2% better. **The pruned networks only contained half of the original weights**. However, the performance on the test set was not significantly improved, **the classification error on the test set was 36.7%**.

An **MLP with a single hidden layer** containing 11 units, i.e. 242 weights plus 17 thresholds was designed. The training was done by a standard Back-Propagation. **The classification error on the test set was 28.7%**. Again the network was pruned by a MAG, and the network was then cut down to 7 hidden units and a total number of weights as low as 45. The classification error on the training set was still 0 while **the error on the test set increased by 1%**.

An **MLP with two hidden layers** containing 16 units in the first layer and 3 in the second, i.e. 322 weights plus 25 thresholds was designed. The training was done by a standard Back-Propagation. **The classification error on the test set was 32.8%**. Again the network was pruned by a MAG. The network was then cut down to 6 hidden units in the first layer, 2 in the second hidden layer and a total number of weights as low as 56. **The classification error on the test set increased to 36.2%**.

The performances of both **CCA** and **GLOCAL** were also tested. For both algorithms the result was the same as for the simple perceptron architecture because the perceptron is able to learn the training set.

7.4.4 Conclusion (*for experiments with the 30/5780 data set*)

It is not possible to make a final conclusion on the basis of these experiments whether a neural network would be a good alternative to traditional methods. Traditional methods make a better performance, with 23.9% classification error on the test set, while the best performing network in these experiments has a classification error on the training set of 28.7%. None of these rates are, however, very impressive and the reason is that the training set is too small to represent the problem. This is not surprising for several reasons: the training set was actually picked so only segments with large areas were members, while the smaller segments had a different distribution of features; it is extremely rare that the whole problem can be represented by only 0.5% of the data unless the problem is trivial.

In chapter 4 it was shown that with a single output, one hidden layer MLP having units with a linear threshold function, the number of training examples being at least 10 times the number of weights multiplied by the number of hidden units and a training error less than 5%, then the expected error on the test set would be about 10% if the test set is drawn from the same distribution. Applying this to the ice data and the MLP with 7 hidden units and 45 weights, the size of the training set should be approximately 3,000 examples. In other cases these results have shown to be rather pessimistic, but it gives an idea of the size of the training set needed to allow one to expect a reasonable test set error (generalisation error).

An additional result from these experiments came from cases where pruning was used. Pruning often removed all weights to a single input unit, thereby telling that this input could be cancelled. Different inputs were cancelled from one network to another, but in general the Minimum intensity and the Normalised 4th order moment were cancelled. This result was confirmed by [Skriver 95]. The most exciting thing about this result is that in future works it might be possible to get more knowledge about the importance of the different features' mutual dependence, by observing how the pruning process elapses.

7.4.5 New Partition of Data

The argument for choosing the small training set was that the $BMLC$ only works properly with a statistical estimate of the features that are central and uncorrelated, and this is only valid for large segments. Neural nets do not take into consideration or make any assumption about the statistical properties of the data, that is one of their strengths[4].

[4]Some would argue that this is not the case, see for example [Geman *et al.* 92]

Therefore it makes no sense not to use the data in a better way. There is no problem in selecting all data as training set for a neural network, but since the essential quality is its ability to generalise, i.e. its performance on unknown data, a test set is needed. So the question is how to split the available data in a meaningful and fair way. What is fair then? No strict rules are given, but different opportunities are often used. What is most often seen is that the data set is split up so the training set contains between 30 and 70%, and the test set contains the rest of the elements. There are also techniques known as cross-validation that allow one to use a larger part of the data for training. An especially popular method is the *leave-one-out* where all data minus one are applied for training (see chapter 4). The disadvantage of this method is that this should be done for each data element in order to get a representative result.

In the following experiments data in the training and test set were chosen at random in such a way that each class in the respective sets contained 50% of all data. No attempt was made to ensure that previous training sets of large segments were included in the new training set. This means that data were split in the following way: a **training set with 2,892 data, (1,724** MY**, 54** NIa**, 347** NIb**, 322** NIc**, 199** WA**, 246** MIX**)** and a **test set with 2,888 data**.

As already mentioned the size of a training set is of vital importance for the performance that can be expected of a network. With a large training set it is often advisable to use an on-line training strategy, off-line can be cruelly slow. In such cases it is of great importance how the patterns in the training set are ordered. A practical way to solve this problem is to *shuffle*$^{\heartsuit}$ the data, this means that the data are presented to the network in a varied order.

7.4.6 Analysis of Data

In order to get an idea of how difficult the problem is given this new normalised training set, two alternative analyses were made.

Parameter plot: In the first analysis each feature for all data was plotted and labelled according to its class. The result was blurred as the points from the different classes were placed on each other. However, it seemed that the points from each class with a crude approximation had a Gaussian distribution. The mean and standard deviations were therefore calculated and the Gaussian function with these features was plotted instead of the first approach. The result is shown in figures 7.3 and 7.4.

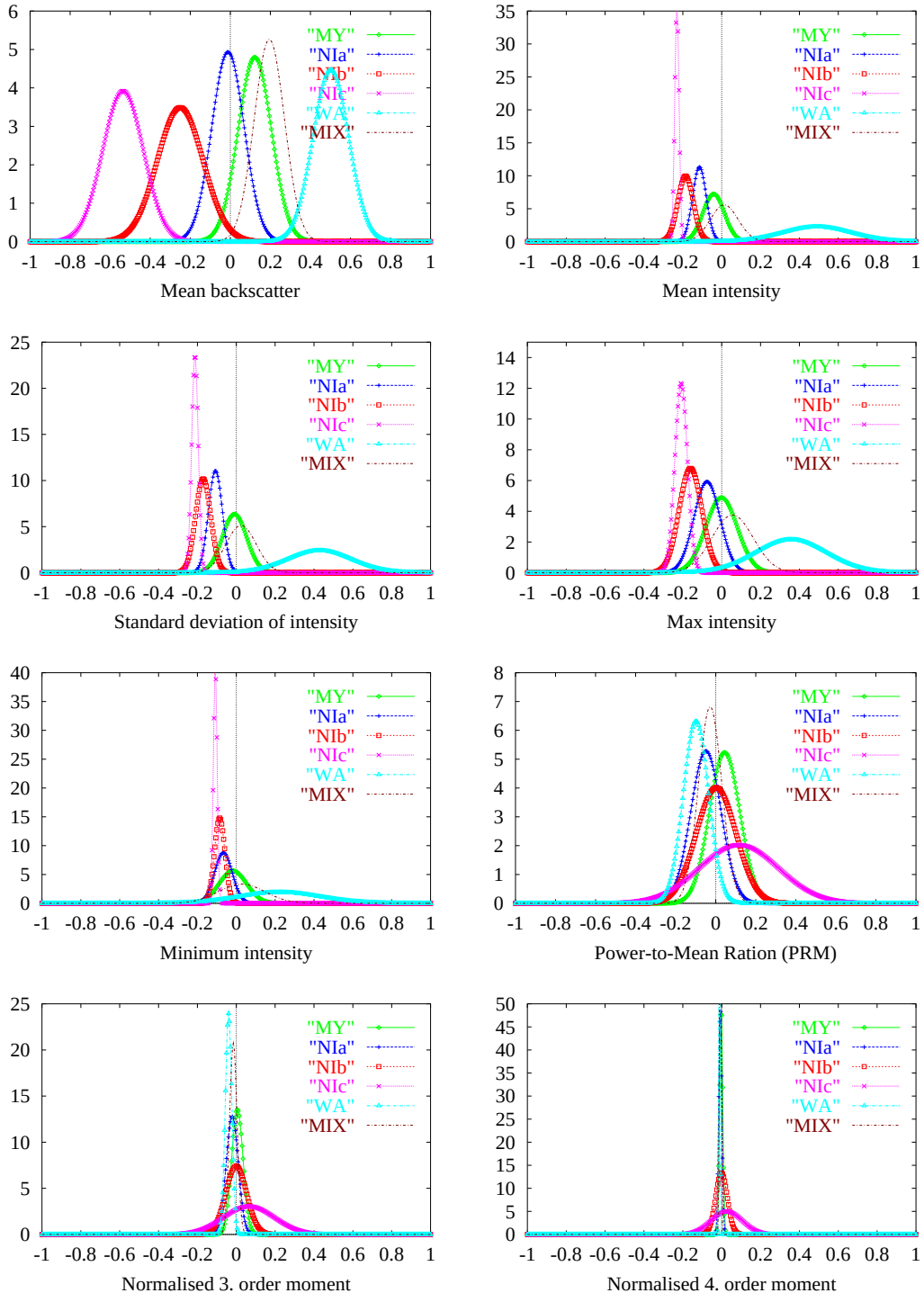


Figure 7.3: The first 8 features for each class displayed as if they had a normal distribution.

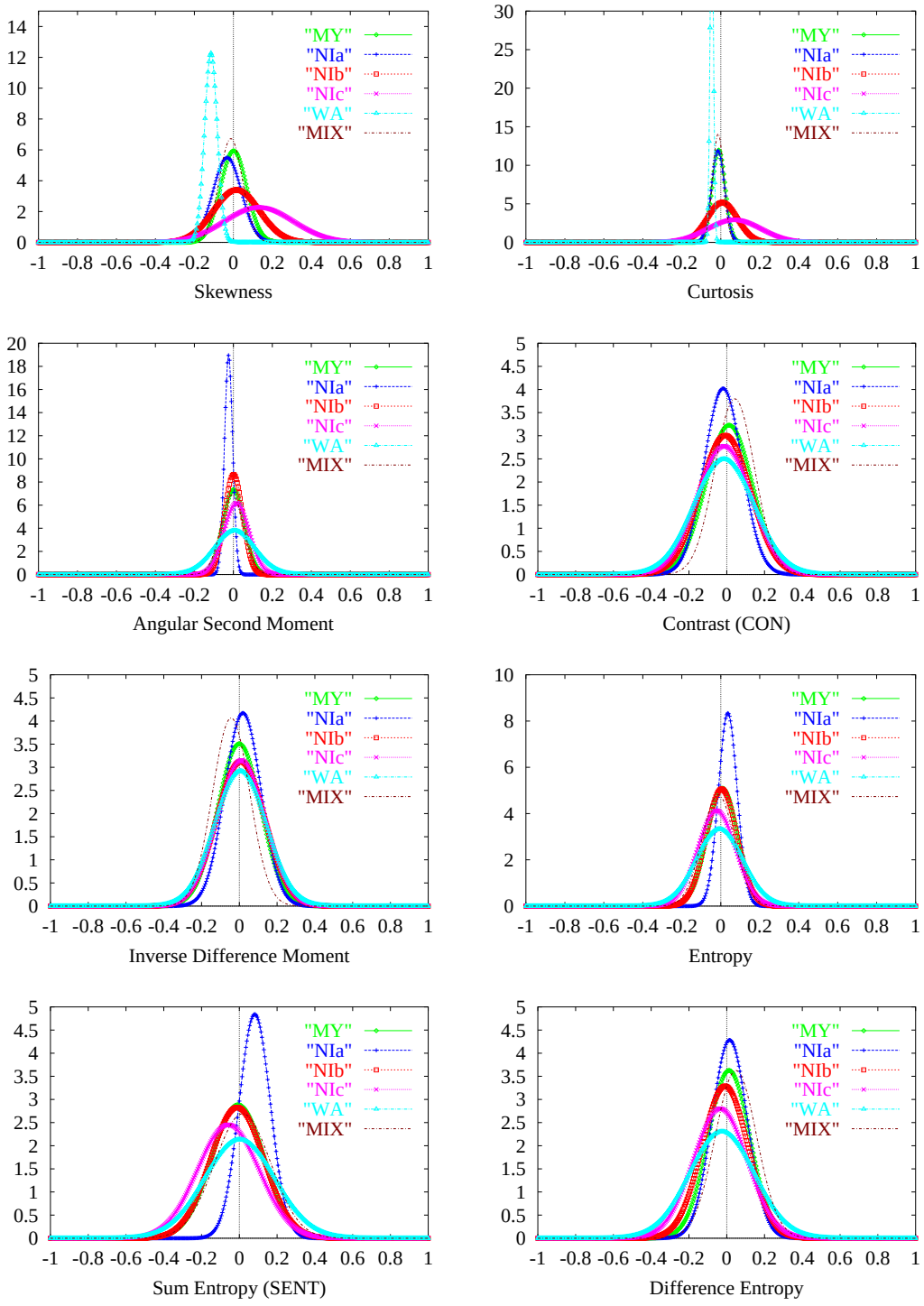


Figure 7.4: The last 8 features for each class displayed as if they had a normal distribution.

This analysis clearly shows that most of the features are so identical that only very little information can be expected. An exception is the *Mean backscatter* feature.

Self-Organised-Map:♡ The last analysis was also an attempt to measure how close the data were to each other. An un-supervised clustering method belonging to a class of neural networks called Self-Organised-Map [SOM 95] was used. Several different networks (number of output units and thereby possible clusters) were constructed and more than 20,000 different Maps were produced. In figure 7.5 the best performing network is shown. Best performing means the map with the lowest quantisation error.

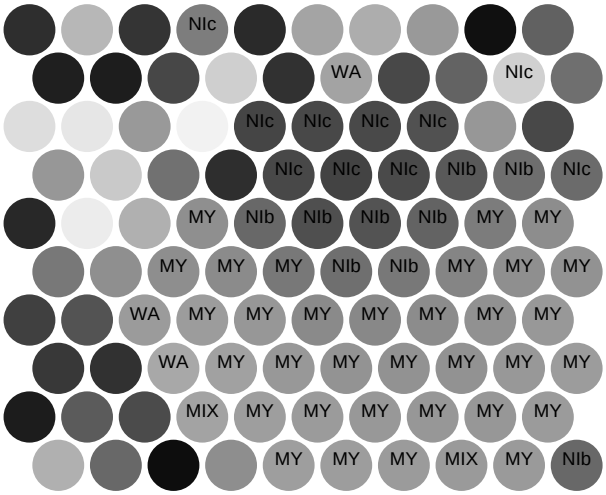


Figure 7.5: The "MAP" - the Self-Organised-Map. *(Note that Class NIa was not able to get its own class.)*

In figure 7.6 illustrations of the *road* that training patterns from each class follow in the Self-Organised Map are shown. The road is a line drawn through the units that give the highest response when the training patterns are presented to the network.

The most striking result from this analysis is that the NIa class is not able to get its own class. The reason is that the few members in this class are distributed in such a way that the majority of responses will come from the MY class that is located in the same area. One more thing should be noted: none of the classes is only visited by its own true members.

The final conclusion to be drawn from these alternative analyses is that the problem defined by the training set is a difficult task and it is impossible to learn the entire training set while it is inconsistent (there are misclassifications or ambiguous elements in the training set). It also clearly illustrates the effect of having a training set where the ratio between the different classes is unequal.

It is not possible to anticipate what the network will learn, but it should at least be able to learn about 60% of the training data. This corresponds to the trivial case where the MY class is always chosen.

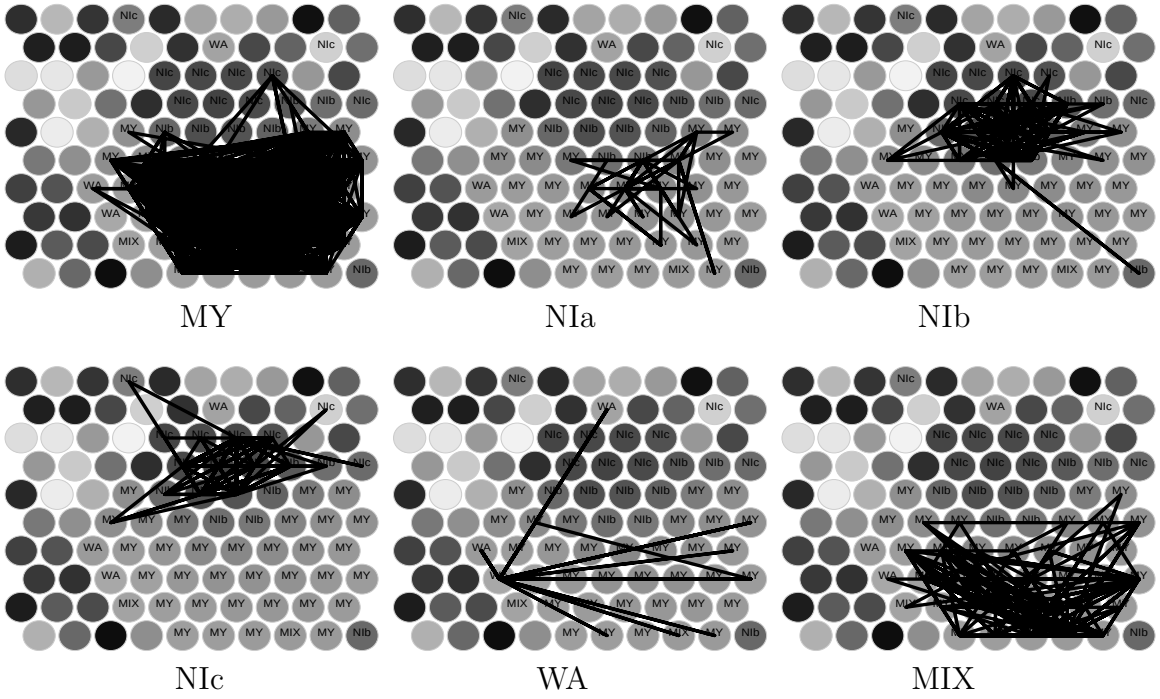


Figure 7.6: The *road* that training patterns follow in the Self-Organised-Map.

7.4.7 Implementation, Experiments and Results (*on the 2892/2888 data set*)

A representative number of the networks designed and implemented is shown in table 7.5.

General comments on the implementations:

The requirement to all neural networks, except the third MLP with one hidden layer, was that the training should continue until the MSE did not show any progress. When pruning was performed, the requirement to the pruning was that it should stop as soon as the network made an increase on the MSE that was larger than 10% of the original network's MSE, or when the number of elements misclassified increased by more than 30, and then use the last weight combination that was able to show a classification error on the training set with no more than a 10% increase.

In all experiments **shuffle** was used on the training set and it had a vital influence on the network's performance and result.

Comments on the implementations:

A **simple perceptron** with 16 inputs and 6 outputs, i.e. 96 weights plus 6 thresholds was designed. The training was done by a standard Back-Propagation. **The classification error on the training set was 9.6%**. This means that the problem described by this training set is no longer linearly separable. This was expected from the analysis of the data. **The classification error on the test set was 11.0%**. The number of weights in the network was then reduced using the Magnitude Pruning algorithm (MAG). The

TYPE	Architecture	Number of Weights	Learning Algorithms	Training Set (Classification Error in %)	Test set (Classification Error in %)
PERCEP	0	96	Std_BP	9.6	11.0
PERCEP	0	50	Pru_OBS	10.4	10.8
MLP	11	242	Std_BP	9.1	11.6
MLP	10	161	Pru_Mag	9.6	12.0
MLP	11	242	Std_BP	10.8	10.8
MLP	16+3	322	Std_BP	8.3	10.7
MLP	16+3	242	Pru_Mag	14.2	14.1
MLP	32+16	1120	SCG_lms	9.6	10.3
MLP	32+16	1120	SCG_Exp	10.3	Not Tested
MLP	32+16	1120	SCG-$\sigma = 0.3$	10.9	Not Tested
CCA	40×1	1756	Quick-Prop	12.4	12.8
GLOCAL	46	782	Std_BP	10.0	11.2
PERCEP	0	96	Std_BP	11.0	10.3

Table 7.5: Performance table for an MLP, the CCA and GLOCAL on the SAR data. *X+Y means X unit in the first hidden layer and Y in the second hidden layer, while* $X \times 1$ *means X hidden layers with one unit.*

pruned networks only contained half of the original weights. However, the performance on the test set was not significantly improved, **the classification error on the test set was 10.4%**.

An **MLP** with a single hidden layer containing 11 units i.e. 242 weights plus 17 thresholds was designed. The training was done by a standard Back-Propagation. **The classification errors on the training set and the test set were then 9.1% and 11.6% respectively**. The number of weights in the network was then reduced using the MAG pruning algorithm. The pruned networks only contained 66% of the original weights. The performance on the test set did show some improvement, the classification error on the test set was 10.7%. Another test with this architecture and learning parameter where the test set was used in an **early stopping** manner gave a **classification error on both the training set and the test set of 10.8%**.

An **MLP with two hidden layers** containing 16 units in the first layer and 3 in the second, i.e. 322 weights plus 25 thresholds was designed. The training was done by a standard Back-Propagation. **The classification error on the training set was 8.3%**. This was the best result ever shown in these experiments on the training set. **The classification error on the test set was more "normal", namely 10.7%**. Again the network was pruned by a MAG. But with the 10% requirement to the pruning no weights were allowed to be pruned. A rather drastic change in the requirement to the pruning was necessary to allow any weight to be pruned. The requirement was therefore changed so the increase in the training set error was not allowed to be larger than 100% of the original network. The network was then cut down to 242 weights, no units were removed. **The classification errors on both the training set and the test set**

were increased to 14.2% and 14.1% respectively.

An **MLP with two hidden layers** containing 32 units in the first layer and 16 in the second, i.e. 322 weights plus 25 thresholds was designed. The training was done by the Scaled Conjugate Gradient method, see [Møller 93a]. Two similar experiments were performed with the only difference that the error functions were the Soft-Monotonic error function and the $MS - \sigma$ error function described in chapter 3. **The classification errors were 10.3% and 10.9%** which is only a minor difference.

A **CCA** using the Quick-Prop learning algorithm was used to automatically build a network. The number of candidates was set to 8 and the maximum number of hidden units allowed was set to 40. All 40 hidden units were inserted and the final network had a **classification error on the training set of 12.4% and a classification error on the test set of 12.8%**.

The **GLOCAL** algorithm using a standard BP learning algorithm was used to automatically build a network. The number of RBF units allowed was set to 40. The initialised perceptron inside GLOCAL was trained until the training set error was 11%. 40 RBF units were then inserted and **the training error was reduced to 10.0%**. On the test set, however, the error was increased as a consequence of inserting the RBF units. The reason is that the RBF units are placed on patterns that are probably misclassified. This was confirmed by varying the number of elements inside a cluster required for inserting an RBF unit. With the requirement that all the elements in a cluster should be at least 0.3% (10 patterns), only 4 RBF units were inserted. But the general trend was still the same: lower training set error and higher test set error.

7.4.8 Conclusion (*for the 2892/2888 data set*)

The best result on the training was performed by the $MLP(16+3)$. The following table shows the training and test confusion matrix for this network. The test confusion matrix for the $BMLC$ is also shown. In order to make the results directly comparable, the values from table 3 are halved, which corresponds to the fact that the test with the $BMLC$ was performed on a test set with 2,892 elements.

The training confusion matrix for the "$MLP(16+3)$" classifier is shown below:

Ice Type	MY	NIa	NIb	NIc	WA	MIX	Errors
MY	1698	21	6	0	2	96	125
NIa	0	6	0	0	0	0	0
NIb	9	23	313	32	0	0	64
NIc	1	0	28	290	0	0	29
WA	0	0	0	0	196	1	1
MIX	16	4	0	0	1	149	21

Table 7.6: Training confusion matrix for the $MLP(16+3)$ *The total number of errors is = 240*

The perfect confusion matrix would be diagonal with 1724 MY, 54 NIa, 347 NIb, 322 NIc, 199 WA, and 246 MIX and the rest of the matrix entries zero. Note that the network is able to classify some of the NIa members, although the number of training patterns is alarmingly small.

The two following tables show the test confusion matrix for the $MLP(16+3)$ and the $BMLC$ respectively.

Ice Type	MY	NIa	NIb	NIc	WA	MIX	Errors
MY	1665	17	15	0	15	116	163
NIa	1	0	0	0	0	0	1
NIb	34	28	315	20	0	0	82
NIc	0	1	15	300	0	0	16
WA	2	0	1	1	175	4	8
MIX	22	8	0	0	8	126	38

Table 7.7: Test confusion matrix for $MLP(16+3)$. *The total number of errors is 308 ($\approx 10.7\%$)*

The confusion matrix for the $BMLC$ with BC & PMR classifier is shown below:

Ice Type	MY	NIa	NIb	NIc	WA	MIX	Errors
MY	1573	110	10	3	1	27	151
NIa	11	34	7	0	0	1	19
NIb	9	78	147	111	1	0	199
NIc	0	0	2	319	0	0	2
WA	89	0	0	0	109	1	90
MIX	217	10	0	0	2	16	229

Table 7.8: Confusion matrix for the $BMLC$ with BC & PMR. *The total number of errors is 690 ($\approx 23.9\%$)*

Comparing these two tables clearly reveals why the neural network performs much better than the $BMLC$: it is simply able to learn to classify classes with a relatively large number of patterns. Further investigations will be necessary in order to disclose whether this is due to the quality of a neural network or merely due to the fact that it shows more patterns. For the overall result this might be of minor interest, the important thing is that the network is able to be trained on a large training set, thereby improving the classification rate.

Another interesting observation is that the type of network given a data set with such a unequal distribution, is not of significant importance for the performance. So **improvement of the classification rate may be obtained by using a simple perceptron**.

7.4.9 Equally Distributed Training Set

The previous training set was biased towards MY since it contains most examples. If one always chooses MY as class, the error will only be $1-\frac{1724}{2892} \approx 40\%$. In order to eliminate this bias two new training sets were constructed, so all classes were more equally represented. The training set was constructed on the basis of the existing training set. In the first case, the elements from the MY class were halved by simply picking the first 862 elements. The last five classes were found by the following formula: $Truncate\frac{862}{\#Class} * \#Class$. This gave a training set containing 4,544 elements, (862 MY, 810 NIa, 694 NIb, 644 NIc, 796 WA, 738 MIX). An MLP with 32+16 hidden units was trained and tested on this training set, the result is shown in table 7.9.

TYPE	Architecture	Number of Weights	Learning Algorithms	Training Set (Classification Error in %)	Test set (Classification Error in %)
MLP	32+16	1120	SCG_Exp	5.6	17.8
MLP	32+16	1120	SCG-$\sigma = 0.3$	6.3	16.3

Table 7.9: Performance table for the MLP(32+16) on the SAR data with equal number of elements in the training set. *X+Y means X unit in the first hidden layer and Y in the second hidden layer.*

In the second case, all classes were simply scaled up so the number of elements was almost equal to the number of elements in the MY class. This gave a **training set containing 10310 elements**, (1724 MY, 1728 NIa, 1735 NIb, 1610 NIc, 1791 WA, 1722 MIX). Again an MLP with 32+16 hidden units was trained and tested, the result is shown in table 7.10 and the confusion matrix in table 7.11.

TYPE	Architecture	Number of Weights	Learning Algorithms	Training Set (Classification Error in %)	Test set (Classification Error in %)
MLP	32+16	1120	Std_BP	2.6	16.0

Table 7.10: Performance table for the MLP (32+16) on the SAR data with equal number of elements in a 10310 elements training set. *X+Y means X unit in the first hidden layer and Y in the second hidden layer.*

Ice Type	MY	NIa	NIb	NIc	WA	MIX	Errors
MY	1604	0	5	5	0	49	59
NIa	6	1728	0	0	9	7	22
NIb	10	0	1705	35	9	7	61
NIc	1	0	25	1570	0	0	26
WA	2	0	0	0	1773	1	3
MIX	101	0	0	0	0	1659	101

Table 7.11: Training confusion matrix for the $MLP(32+16)$ *The total number of errors is = 272*

7.4.10 Conclusion (*equally distributed training set*)

The performance on this training set shows that the network of the size MLP(32+16) was able to learn the training set better than what has been seen before (compare the result in table 7.10 with 7.5 and 7.9). The reason is that most elements from each class are now learned (see the confusion matrix). The classification error on the test set is on the other hand relatively high. This is caused by the fact that the new training set has only little influence on the performance of the MY class, cf. table 7.6. It seems that some of the other classes are able to capture classification areas from MY, because they are now more strongly represented. Although the other classes are able to get a relatively low classification error, the major contributor to the classification error is still the MY class, and since the test set has not been changed, the performance could not be expected to change either.

7.5 Classification on the Basis of Pixel Values

As an alternative to the features, the segmentation algorithm also produces a 9x9 pixel window placed at the centre of the segment which contains the raw data. The reason why a 9x9 window was chosen was a wish to be able to present the smallest segment without filling artificial values. The total number of patterns in the data set was 4,740. It was split into a training set with 2,368 patterns and a test set with 2,372 patterns. In the training set each class was represented in the following way: **1,420** MY**, 52** NIa**, 280** NIb**, 243** NIc**, 168** WA**, 205** MIX.

It could be expected that these data will give a better result than the features, because the features are all linear combinations of the data, and thereby they do not take full advantage of the neural network's ability to handle non-linearities. On the other hand, neural networks have shown some difficulties in extrapolating information, and it might be necessary to enforce some bias [Geman *et al.* 92]. Also, the 9 x 9 window in general represents a much smaller number of pixels than the original segments.

For each of the two network types $PERCEP$ and $MLP(25)$ about 20 different networks were implemented. The best results from these experiments are shown in table 7.12.

TYPE	Architecture	Number of Weights	Learning Algorithms	Training Set (Classification Error in %)	Test set (Classification Error in %)
PERCEP	0	486	Std_BP	30.0	30.0
MLP	25	2175	Std_BP	10.4	30.0
MLP	25	2175	Pru_Mag	22.0	25.6

Table 7.12: Performance table for the MLP on the RAW SAR data.

The perceptron was able to recognise 99.5% of the MY patterns and 99.2% of NIc, while none of the other classes was ever correctly classified. The first MLP network in table 12 was able to learn to classify patterns from all classes, but the price seems to be

overfitting. This was confirmed by the second MLP in table 7.12 which shows that by stopping the training before the learning error was at its lowest, one gets a better test set error.

7.5.1 Conclusion *(on the pixel data set)*

Based on this rather weak foundation it seems that a neural network will be able to work with raw pixel values as input. But as the number of input parameters increases, the number of weights and units inside the network also increases and therefore it can be expected that a larger data set is needed. This might very well mean that the training and optimisation procedure becomes infeasible.

7.6 Conclusion of the Preliminary Investigation

Experiments showed that almost any kind of neural network, using a Standard Back-Propagation (Std_BP) learning algorithm for minimising the Mean Square error, is able to perform better than the Bayesian Maximum Likelihood Classifier ($BMLC$). The reason is that the neural network is able to use a larger training set than the $BMLC$. It turned out that for a given data set with an unequal distribution available for these experiments, a simple perceptron would improve the classification rate compared to the traditional $BMLC$. An attempt to compensate for the unequal distribution by making training sets with equal elements in each class showed that the neural network could be trained to learn the training set with a classification error as low as 2.6%. Its generalisation ability was, however, not improved. It also seems that the choice of error function and learning algorithm was less important. Finally the possibility to work on the raw pixel values seems to be there, but it is likely to become infeasible.

To do the Bayesian Maximum Likelihood Classifier method justice it should be mentioned that the one that is used at the Electromagnetics Institute (EMI), Technical University of Denmark is probably not the optimal one. A larger training set, balanced between the desire of many training data and a growing variance of the estimated features and the correlations between the features in small segments, would probably improve the performance. A better selection of the features used, such as Karhunen-Loeve transformation, would probably improve the result further thereby giving a better foundation for a comparison. However, neural networks could also very well benefit from a Karhunen-Loeve transformation and the networks described in this preliminary investigation are probably also not the best.

The main conclusion of this preliminary investigation is that it would be profitable to apply a neural network in order to classify ice types from the SAR-data. There is, however, no doubt that further research in collaboration with people having expert knowledge of SAR-data is needed, before the results will have a more practical value.

Chapter 8

Conclusion

This chapter will go more thoroughly into a conclusion and highlight some of the results that have been achieved. It starts by putting the work in perspective in relation to the three educational subgoals that are usually formulated in the educational plan. Then the focus will be on the new things that this thesis has contributed with. Suggestions on future work will also be given and finally there will be some remarks from the author on his experiences from this project and his personal view on the perspectives of neural networks.

8.1 Introduction

The **overall purpose** of the project "Automated Design of Neural Network Architecture for Classification" **was to investigate the possibility of finding a method or strategy for automatically building neural networks so the network will be able to classify objects in radar signals**. The type of network that seemed most appropriate for this task was a feed-forward network and the investigation was therefore constrained to this type of network. The previous chapters have all dealt with topics that were directly related to or in some way were important in order to achieve this goal. The focus has been on methods to improve the performance of existing architectures (i.e. architectures that are initialised by a good academic guess) and automatically building neural networks. Several different approaches have been used and discussed, but also more fundamental issues like *learning* and *generalisation* have been analysed. The **overall conclusion** of this investigation is that **there are several possible methods or strategies for automatically building neural networks that are able to solve the task**.

8.2 Were the Goals Achieved?

In order to support the overall conclusion the work performed during the project will be compared to the three educational subgoals **Research level**, **Development level** and **Application level** as formulated in the educational plan for the project, see chapter 1 or Appendix B.

8.2.1 Research Level

Goal: Establish a high level of knowledge of methods and techniques for building neural network classifiers. The success of a neural network is strongly dependent on finding the right network architecture. The emphasis of this research will be on automatic generation of network architecture and methods to improve the performance of existing architectures.

This goal was achieved first by a thorough analysis of some of the basic elements like learning and generalisation in order to establish the foundation for further work. It was shown that the use of the mean square error functions applied in the training phase of a neural network is not necessarily the best to use for classification tasks. Error functions with *soft-monotonic* functions will make the classification more correct and the use of an error function that includes an *offset* term like the *MS-*σ error function eliminates the effect of local minima. Further the philosophy of Ockham's Razor was confirmed: the network that is able to learn the training data in a satisfactory way using the smallest number of parameters, is the network with the largest probability of a good generalisation ability, see chapters 3 and 4. Secondly and most important were the analysis, development and test of methods and strategy for automatically building neural networks in chapters 5 and 6. Here it was clearly demonstrated that the use of methods like Early Stopping, Pruning and construction algorithms, especially the newly developed GLOCAL, made it possible both to improve the performance of existing architectures and to generate network architectures automatically.

8.2.2 Development Level

Goal: Methods and algorithms for automatically generating neural network architectures or optimising existing neural networks are to be developed and implemented and then used to build an artificial neural network which is able to perform classification of various radar sources. The neural network must accept preprocessed data from a Synthetic Aperture Radar (SAR) and perform an analysis which establishes a classification of different objects (ice types).

The methods and algorithms for optimising existing neural networks or automatically generating neural network architectures that were developed and implemented during the work were described in chapters 5 and 6. These methods were applied to improve/build neural networks that were able to perform an analysis of signals from a Synthetic Aperture Radar which establishes a classification of different ice types, see chapter 7.

8.2.3 Application Level

Goal: It is expected that the developed algorithms and methods will have such a general structure that they will be useful for a broad spectrum of other applications. Examples of such applications go from Classification of Radar Targets, Electronic Data Interchange (EDI), Data Fusion, to Analysis of formatted messages to or from a C3I system.

Besides the SAR data all the methods and algorithms have been tested on several different classification problems and can easily be adapted to other classification problems, so nothing prevents the use of these methods and algorithms for a broad spectrum of other applications.

8.3 To What has this Work Contributed?

The major contributions of the work presented in this thesis are:

- Three new error functions were introduced: the Exponential and the Variance error functions, based on the idea that soft-monotonic error functions produce a more correct classification, and the MS-σ that makes it possible to apply the offset approach in connection with second order training algorithms like the Scaled Conjugate Gradient algorithm. The offset approach is able to speed up the learning but what is more important it prevents the network's output units from being trained to saturation.

- In connection with the measurement of a network's capacity the *VC-dimension* was described. The $\mathcal{VC}_{dim}$ is in general unknown, an exception is the linear classifier. It was shown that many of the construction algorithms that have been developed during the last few years have the same $\mathcal{VC}_{dim}$ as a linear classifier.

- An analysis of some very popular pruning methods - *Magnitude Based Pruning*, *Optimal Brain Damage* and *Optimal Brain Surgeon* used to improve the generalisation ability of a neural network with a predetermined architecture containing many parameters - showed that no single method was the best all the time so it does not seem to be important which method is used for pruning as long as one method is used.

- For real-life applications where the complexity might be very large, the following pruning scheme was suggested. Start with MAG until the network has a size that will allow the use of OBS, and then make an effort to find a "good" local minimum in order to give OBS good start conditions. When such a minimum is found OBS should be used to do the rest of the pruning. When OBS stops the network should be retrained in order to get a minimum and OBS should be started again.

- A survey of several construction algorithms including pseudo-algorithms and graphical illustrations was given.

- A new construction algorithm that combines units with both types of transfer functions in the hidden layer. This algorithm called *the GLOCAL Algorithm* was described in detail. The experience with GLOCAL is encouraging so far, it has an easy analytic traceability due to its modular construction and clustering nature, and its performance in experiments and tests is comparable to the performance of networks constructed on a more classical approach i.e. where the design is basically made on a qualified and academic guess and some rules of thumb that are repeated until success.

- Experiments showing that almost any kind of neural network, using a Standard Back-Propagation (Std_BP) learning algorithm for minimising the Mean Square error, is able to perform better than the Bayesian Maximum Likelihood Classifier ($BMLC$).

8.4 Suggestions on Future work

Throughout the thesis several open questions were left unanswered. In order to answer these questions further work will be needed in the areas of:

Combining different methods to improve predetemined architectures. The experiment showed that Weight Decay along with pruning methods that require training and retraining is a good idea. Several successful tests are shown in [Finnoff *et al.* 93] where a combination of Early Stopping is used in connection with various regularisation and pruning methods, including Weight Decay and OBD.

Optimal stop criteria for pruning methods. Different stop criteria may be used, it is e.g. often seen that an increase in the learning error after retraining is used or as far as OBS is concerned the size of the saliency (expected increase in error) is used. An obvious way is instead to employ a validation set and stop the pruning when the validation ratio exceeds a threshold as described in the paragraph on Early Stopping.

Supervised Clustering. In the further development of GLOCAL future work will include new, faster and better cluster strategies and algorithms and an analysis of how the number of RBF units can be controlled when the perceptron is trained with Early Stopping.

Metric and Dimensionality. Most methods that are based on a local measurement assume that the metric is the Euclidian distance (2-norm). This is not necessarily the best, in the supervised cluster case the optimal solution would be to find a metric, probably a nonlinear weighted distance measure, so that two points from different classes having a distance between them at zero in the Euclidian norm would be infinite. This would open the way to the use of a large number of well-known cluster methods. Another question is how to deal with higher dimensionality, which also has a certain influence on the metric to be used. Discussions on and illustrations of these problems can be found in [Duda and Hart 73], [Schalkoff 92], [4] and [6]. It should be stressed that the answers are not trivial, but indeed very important for the overall performance, perhaps the most important.

Measurement of the generalisation ability. One of the everlasting questions is how to measure the generalisation ability of different learning machines. Until a measurement of the generalisation ability is found there is no quality measurement for preferring one method to the other, except for practical reasons.

8.5 Author's Remarks

My original opinion on neural networks was that they were some of the most fascinating and promising techniques available for handling tasks of nonlinear nature. During this project my fascination of and belief in neural networks' potential have not decreased; I still think that a neural network is a technique which can by applied to many applications.

My view on finding methods that automatically build networks has, however, changed. My first naive goal was to find a method that could help designers or engineers with only

little knowledge of neural networks build networks that would be able to solve their classification tasks successfully. Indeed the work in this thesis has demonstrated that it is possible to find very good methods for automatically building neural networks, including both methods to improve predetermined architectures and constructions algorithms. It has also turned out that the number of learning/method parameters that have to be determined in advance is so large and that these are so important for the overall performance that in most cases it requires a considerable amount of skills and knowledge of neural networks to build a successful network.

Once I heard D.E. Goldberg[1] talk about the similarities or rather dissimilarities between research and development today and the work performed during the 4-year period it took the Wright brothers to build and fly *Flyer I*. The spirit from this talk - the way I hear it - was: They were able to fly and manoeuvre the machine almost as they wanted, without knowing everything about the mathematics that it takes to build an aeroplane today, but it did not keep them from trying. I guess the same goes for neural networks, we can use them and control them, to our advantage, but still we only know little about the mathematics. When it comes to automatic methods I think that they can help make better network solutions, but still they should only be used by people with major knowledge of neural networks just as it takes skilled personnel to fly and to get success. There is still a lot of work to be done, so jump on board!

[1]D.E. Goldberg (author of the first GA book) presented a "Genetic Algorithm Tutorial" at the University of Aarhus on January 13, 1995.

Bibliography

[]GENERAL DESCRIPTION OF NEURAL NETWORKS

[DARPA 88] *DARPA Neural Network Study.* (1988), AFCEA International Press.

[Funahashi 89] Funahashi, K. (1989), *On the Approximate Realization of Continuous Mappings by Neural Networks.* Neural Networks, 2, 183-192

[Gibson and Cowan 90] Gibson, G.J. and Cowan, C.F.N. (1990), *On the Decision Regions of Multilayer Perceptrons*, Proceedings of the IEEE, vol 78, No. 10, October 1990.

[Hebb 49] Hebb, D. (1949), *The Organization of Behavior*, New York: John Wiley and Sons

[Hertz *et al.* 91] Hertz, J., Krogh, A. and Palmer, R. (1991), *Introduction to the Theory of Neural Computation.* Addison Wesley.

[Huang and Lippmann 87] Huang, W.H. and Lippmann, R.P. (1987), *Neural net and traditional classifiers*, In Neural Information Processing Systems (Denver 1987).

[Kohonen 84] Kohonen, T. (1984) *Self-Organization and Associative Memory*, Berlin: Springer-Verlag.

[Lapedes and Farber 88] Lapedes, A. and Farber R. (1988), *How Neural Nets Work.* In Neural Information Processing Systems, ed. D.Z. Anderson, pp 442-456. New York: American Institute of Physics.

[Le Cun 93a] Le Cun, Y. (1993), *Efficient Learning & Second-Order Methods - A Tutorial at NIPS*93.* Notes from NIPS 93, Denver MARRIOT City Center Denver, Colorado, November 29, 1993. 76 pages.

[Lippmann 87] Lippmann, R.P. (1987), *An Introduction to Computing with Neural Nets.* IEEE ASSP Magazine, April 1987, pp. 4-22.

[Lutzy *et al.* 93] Lutzy, O. and Dengel, A. (1993), *A Comparison of Neural Net Simulators* IEEE Expert, August 1993, pp. 43-51.

[McCulloch and Pitts 43] McCulloch, W. and Pitts, W. (1943) *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics. 7, pp. 115-133.

[Michie *et al.* 94] Michie, D., Spiegelhalter, D.J. and Taylor, C.C. (1994), *Machine Learning, Neural and Statistical Classification*, Ellis Horwood series in Artificial Intelligence, 289 pages.

[Minsky and Papert 69] Minsky, M. and Papert, S. (1969), *Perceptrons*, Cambridge MA: MIT Press. 258 pages.

[Nilsson 89] Nilsson, N.J. (1965), *Learning Machines - Foundations of trainable pattern-classifying systems*. McGraw-Hill Book Company. 137 pages.

[Pao 89] Pao, H. (1989), *Adaptive Pattern Recognition and Neural Networks*. 1 - 327. Addison Wesley.

[Prechelt 94] Prechelt, L. (1994), *Proben 1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules.* Found on connectionist ftp list.

[Rosenblatt 62] Rosenblatt, F.(1962), *Principles of Neurodynamics*. New York: Spartan.

[Ripley 94] Ripley, B.D. (1994), *Neural Networks and Related Methods for Classification*. J. R. Statist. Soc B(1994) 56, No. 3, pp. 409-456.

[Rumelhart *et al.* 86] Rumelhart, D. E. and McClelland, J. L. and the PDP Research Group (1986), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Volume 1: Foundations. pp. 1 - 547, Cambridge: MIT Press.

[Rumelhart et al. 86] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986), *Learning Internal Representations by Error Propagation*, Nature 323, pp. 533-536.

[Scalettar and Zee 88] Scalettar, R. and Zee, A. (1988), *Emergence of Grandmother Memory in Feed Forward Networks: Learning with Noise and Forgetfulness*. In Connectionist Models and Their Implications: Readings from Cognitive Science, eds. D. Waltz and J.A. Feldman, Chapter 11, pp 309-332. Norwood: Ablex.

[Schalkoff 92] Schalkoff R. J. (1992), *Pattern Recognition: Statistical, Structural and Neural Approaches*, John Wiley and Sons, Inc. 1992.

[Sejnowski and Rosenberg 87] Sejnowski, T.J and Rosenberg, C.R. (1987), *Parallel networks that learn to pronounce English text*, Complex Systems, Vol. 1, pp. 145-168.

[SOM 95] Kohonen, T. *et al.* (1995), *Manual for the Self-Organised Map Program Package (SOM-PAK), version 3.1 (April 7, 1995)*, Helsinki University of Technology, Laboratory of Computer and Information Science. Available from the Internet site *cochlea.hut.fi* in directory */pub/som_pak*.

[SNNS 94] Zell, A. *et al.* (1994), *Stuttgart Neural Network Simulator User Manual Version 3.3* Report No. 3/94 from University of Stuttgart, Institute for Parallel and Distributed High Performance System (IPVP). Available from the Internet site *ftp.infomatik.uni-stuttgart.de* in directory */pub/SNNS*.

[Thrun *et al.* 91] Thrun, S.B. and 23 co-authors (1991), *The MONK's Problems - A performance comparison of different learning algorithms*, CMU-CS-91-197 Carnegie-Mellon U. Department of Computer Science Tech Report.

[Widrow and Stearns 85] Widrow, B. and Stearns, S.D. (1985), *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, NJ.

[]RADIAL BASIS FUNCTION NETWORKS

[Broomhead and Lowe 88] Broomhead, D. and Lowe, D. (1988), *Multi-variable Interpolation and Adaptive Networks*. RSRE Memo No. 4148. Royal Signals and Radar Establishment, Malvern.

[Hartman *et al.* 90] Hartman, E.J., Keeler, J.D. and Kowalski, J.M. (1990), *Layered Neural Networks with Gaussian Hidden Units as Universal Approximations*. Neural Computation 2, pp. 210-215.

[Moody and Darken 89] Moody J.E. and Darken C.J. (1989), *Fast learning in network of locally-tuned processing units*, Neural Computation 1, pp. 281-294.

[Musavi *et al.* 92] Musavi, M.T., Ahmed, W., Chan, K.H., Faris, K.B. and Hummels, D.M. (1992), *On the Training of Radial Basis Function Classifiers*, Neural Networks, Vol 5, pp. 595-603, 1992.

[Poggio and Girosi 90a] Poggio, T. and Girosi, F. (1990), *Networks for Approximation and Learning*. Proceedings of the IEEE, Vol 78, No. 9, September 1990, pp. 1481 -1497.

[Poggio and Girosi 90b] Poggio, T. and Girosi, F. (1990), *Extensions of a Theory of Networks for Approximation and Learning: dimensionality reduction and clustering*. A.I. Memo No.1167, C.B.I.P. Paper No. 44. Massachusetts Institute of Technology. April 1990. 17 pages.

[Powell 87] Powell M.J.D (1987). *Radial basis functions for multivariant interpolation: a review*. In Algorithms for Approximation (eds. J.C. Mason and M.G. Cox), pp. 143-167. Oxford Clarendon Press.

[]LEARNING AND ERROR FUNCTIONS

[Battiti 92] Battiti, R.(1992), *First and Second-Order Methods for Learning: between Steepest descent and Newton's Method*, Neural Computation, Vol. 4 (2), pp. 141-167.

[Buntine and Weigend 91a] Buntine, W. and Weigend, A. (1991), *Calculating Second Derivatives on Feed-Forward Networks*, submitted to IEEE Transactions on Neural Networks.

[Buntine and Weigend 91b] Buntine, W.L. and Weigend, A.S. (1991), *Bayesian Back-Propagation*, Complex Systems, Vol. 5, pp. 603-643.

[Depenau and Møller 95] Depenau, J. and Møller, M. (1995), *The LMS-σ Error Function*, Proceedings from the World Congress on Neural Networks, Vol I, pp. 614-617, Washington 1995.

[Hampshire 92] Hampshire, J.B. (1992), *A Differential Theory of Learning for Statistical Pattern Recognition with Connectionist Models*, Ph.D. Thesis, School of Computer Science, Carnegie Mellon University.

[Hampshire and Waibel 90] Hampshire, J.B. and Waibel, A.H. (1990), *A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks*, IEEE Transactions on Neural Networks, Vol. 1, No. 2, pp. 216-228.

[Hornik *et al.* 89] Hornik, K., Stinchcombe, M. and White, H. (1989). *Multilayer Feedforward Networks Are Universal Approximators*. Neural Networks 2, pp. 359-366.

[Johansson et al. 91] Johansson, E. M., Dowla, F.U. and Goodman D.M. (1991), *Backpropagation Learning for Multi-Layer Feed-Forward Neural Networks Using the Conjugate Gradient Method*, International Journal of Neural Systems, Vol. 2, No. 4, pp. 291-301.

[Kocabas 91] Kocabas, S. (1991), *A review of learning*, The Knowledge Engineering Review, Vol 6:3, 1991, pp. 195-222.

[Korning 94] Korning, P.G. (1994) *Training of Neural Networks by means of Genetic Algorithms Working on very long Chromosomes* Daimi PB-486, Computer Science Department, Aarhus University.

[Leshno *et al.* 93] Leshno, M., Lin, V.Y., Pinkus, A. and Schocken, S. (1993), *Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function*. Neural Networks, Vol 6, pp. 861-867.

[Makram-Ebeid et al. 89] Mackram-Ebeid, S., Surat, J.A. and Viola, J. (1989), *A Rationalized Backpropagation Learning Algorithm*, In proceedings of the International Joint Conference on Neural Networks, Washington 1989, Vol. 2, pp. 373-380, New York: IEEE.

[Morgan and Bourlard 90] Morgan, N. and Bourlard, H. (1990), *Generalization and parameter estimation in feedforward nets: some experiments*, In Neural Information Processing Systems II, Ed. D.S. Touretsky, Morgan Kaufmann, NIPS II, pp. 630-637.

[Muselli 95] Muselli, M. (1995), *Is Pocket Algorithm Optimal*, Lecture Notes in Artificial Intelligence 904, Computational Learning Theory, Second European Conference, EuroCOLT'95. Barcelona, Spain, March 1995. Proceedings. Springer. Paul Vitanyi (Ed). pp. 287-297.

[Møller 90a] Møller, M. (1990), *CM Algoritmen*, Masters Thesis in Danish, Daimi IR-95, Computer Science Department, Aarhus University.

[Møller 90b] Møller, M. (1990), *Learning by Conjugate Gradients*, In Proceedings of the Sixth International Meeting of Young Computer Scientist, LNCS 464, Springer Verlag, New York, pp. 184-195.

[Møller 93a] Møller, M. (1993), *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Neural Networks, June, Vol. 6, No. 4, pp. 525-533.

[Møller 93b] Møller, M. (1993), *Supervised Learning on Large Redundant Training sets*, International Journal of Neural Systems, Vol. 4, No. 1, pp. 15-25.

[Møller 93c] Møller, M. (1993), *Exact Calculation of the Product of the Hessian Matrix of Feed-Forward Network Error Functions and a Vector in $O(N)$ Time*, Technical Report, Daimi PB-432, Computer Science Department, Aarhus University.

[Møller 93d] Møller, M. (1993), *Adaptive Preconditioning of the Hessian Matrix*, submitted to Neural Computation.

[Møller 93e] Møller, M. (1993), *Efficient Training of Feed-Forward Neural Networks*. Ph.D. Thesis, Daimi PB-464, Computer Science Department, Aarhus University.

[Møller and Depenau 94] Møller, M. and Depenau, J. (1994), *Soft-Monotonic Error Functions*, Proceedings from the World Congress on Neural Networks, Vol 3, pp. 444-449, San Diego 1994.

[Schiffmann *et al.*92] Schiffmann, W. Joost, M. and Werner R. (1992), *Synthesis and Performance Analysis of Multilayer Neural Network Architectures.* Technical Report 16/1992 from University of Koblenz, Institute fur Physics.

[Schmidt and Stidsen 94] Schmidt, M. and Stidsen, T. (1995), *Using Genetic Algorithms to Train Neural Networks using weight sharing, weight pruning and unit pruning*, Computer Science Department, Aarhus University.

[Seber and Wild 89] Seber, G.A.F. and Wild, C.J. (1989), *Non-linear Regression*, John Wiley and Sons, New York.

[Solla et al. 88] Solla, S.A., Levin, E. and Fleisher, M. (1988), *Accelerated Learning in Layered Neural Networks*, Complex Systems, Vol. 2, pp. 625-639.

[]NEURAL NETWORK AND THE QUESTION OF GENERALISATION

[Baum and Haussler 89] Baum, E.B. and Haussler, D. (1989), *What Size Net Gives Valid Generalization?* Neural Computation 1, pp. 151-160.

[Blumer *et al.* 86] Blumer, A., Ehrenfeucht, A., Haussler, D. and Warmuth, M. (1986,) *Learnability and the Vapnik-Chervonenkis Dimension*. Journal of the ACM, Vol. 36, No. 4, October 1989, pp. 929-965.

[Bottou *et al.* 94] Bottou, L., Cortes, C. and Vapnik, V. (1994). *On the Effective VC-Dimension*. Found on connectionist ftp list. 12 pages.

[Cover 65] Cover T.M. (1965), *Geometrical and Statistical Properties of Linear Inequalites with Applications in Pattern Recognition*, IEEE Transactions on Electronic Computers 14, pp. 326-334.

[Denker *et al.* 87] Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L. and Hopfield, J. (1987), *Large Automatic Learning, Rule Extraction, and Generalization*, Complex Systems 1, pp. 877-922.

[Fogel 91] Fogel, D.B. (1991), *An Information Criterion for Optimal Neural Network Selection*. IEEE Transactions on Neural Network, Vol. 2. No. 3. September 1991.

[Geman *et al.* 92] Geman, S., Bienenstock, E. and R. Doursat (1992), *Neural Networks and the Bias/Variance Dilemma*. Neural Computation 4, pp. 1-58.

[Girard 89] Girard, D.A. (1989), *A Fast 'Monte-Carlo Cross-Validation' Procedure for Large Least Squares Problems with Noisy Data*, Numer. Math., Vol. 56, pp. 1-23.

[Hinton 89] Hinton, G. (1989), *Connectionist Learning Procedures*, Artificial Intelligence, Vol. 40, pp. 185-234.

[Judd 87] Judd, J.S. (1987),*Complexity of connectionist learning with various node functions*, COINS Technical Report 87-60, University of Amherst, Amherst, MA.

[Le Cun 89] Le Cun, Y. (1989). *Generalization and Network Design Strategies*, In Connectionism in Perspective, Eds. R. Pfeifer, Z. Schleter, F. Fogelmann and L. Steels, Zurich, Elsevier.

[MacKay 91b] MacKay, D.J.C. (1991), *A Practical Bayesian Framework for Back-Prop Networks*, Neural Computation, Vol. 4, No. 3, pp. 448-472.

[MacKay 92] MacKay, D.J.C. (1992), *Information-Based Objective Functions for Active Data Selection*, Neural Computation, Vol. 4, pp. 590-604.

[MacKay 91] MacKay, D.J.C. (1991), *Bayesian Interpolation*, Neural Computation, Vol. 4, No. 3, pp. 415-447.

[Moody 92] Moody, J.E. (1992), *The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems*, In Neural Information Processing Systems, Ed. Cowan and Giles, Morgan Kaufmann, Vol. 4.

[Moody 94] Moody, J.E. (1994), *Prediction Risk and Architecture Selection for Neural Networks*, Found on connectionist ftp list. To Appear in From Statistics to Neural Networks: Theory and Pattern Recognition Applications, V. Cherkassky, J.H. Friedman and H. Wechsler (eds.), NATO ASI Series F, Springer-Verlag.

[Tishby *et al.* 88] Tishby, N., Levin, E. and Solla, S.A. (1989), *Consistent Inference of Probabilities in Layered Neural Networks: Predictions and Generalization*, IJCNN International Joint Conference on Neural Networks. Vol II. pp. 403-409.

[Rasmussen 93] Rasmussen, C.E.. (1993), *Generalisation in Neural Networks*. Master Thesis from Electronics Institute, Technical University of Denmark, Lyngby, 1993. 83 pages 83.

[Rissanen 84] Rissanen, J. (1984), *Universal Coding, Information , Prediction, and Estimation*, IEEE Transactions on Information Theory, Vol. 30, No. 4, pp. 629-636.

[Sporrin 95] Sporring, J. (1995), *Statistical Aspects of Generalisation in Neural Networks*. Master Thesis from Department of Computer Science, University of Copenhagen, Denmark. 54 pages4.

[Tishby 95] Tishby, N. (1995), *Statistical Physics Models of Supervised Learning*, The Mathematics of Generalisation, Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning, Ed. D.H. Wolpert. pp. 215-242. Addison-Wesley.

[Valiant 84] L.G. Valiant (1994) *A Theory of the Learnable*. Communications of the AMC. Vol. 27. No. 11, November 1984.

[Vapnik and Chervonenskis 71] Vapnik, V.N. and A.YA. Chervonenkis, A.YA. (1971), *On the Uniform Convergence of Realative Frequencies of Events to Their Probabilities*. Theory Prob. Appl. 16. pp. 264-280.

[Vapnik 82] Vapnik, V.N. (1982). *Estimation of Dependences Based on Empirical Data*. Berlin: Springer-Verlag.

[Vapnik 92] Vapnik, V.N. (1992). *Principles of Risk Minimization for Learning Theory* Advances in Neural Information Processing Systems IV (Denver 1992). ed. J.E.Moody et al., 831-838. San Mateo: Morgan Kaufmann.

[Vapnik *et al.* 94] Vapnik, V., Levin, E. and LeCun, Y. (1994). *Measuring the VC-Dimension of a Learning Machine*, Neural Computation, Vol. 6, pp. 851-876.

[Wan 90] Wan, E.A. (1990), *Neural Network Classification: A Bayesian Interpretation*, IEEE Transactions on Neural Networks, Vol. 1 (4), pp. 303-305.

[Wieland and Leighton 87] Wieland, A. and Leighton, R. (1987), *Geometric analysis of neural networks capabilities* In proc. 1st Int. Conf. on Neural Nets, IEEE, San Diego, Ca, June 1987, Vol. 2.

[Widrow 87] Widrow B (1987), *ADALINE and MADALINE*, Plenary Speech, Vol 1 Proc. IEEE 1st Intl. Conf. on Neural Networks, San Diego, Ca, pp. 143-158.

[Wolpert 95] Wolpert, D. (1995), *The Mathematics of Generalization.* Proceedings of the SFI/CNLS Workshop on Formal Approaches to Supervised Learning. Addison Wesley: pp. 115 - 162.

[]NETWORK ARCHITECTURES

[Alpaydin 91] Alpaydin, E. (1991): *GAL : Networks that grow when they learn and shrink when they forget*, TR 91-032, International Computer Science Institute. May 1991.

[Arentoft 95] Arentoft, M. and Flink, M. (1995), *Optimising of net-structure in feed-forward network by means of genetic algorithms.* Report to Neural Network Studygroup (in Danish), Computer Science Department, Aarhus University.

[Carpenter and Grossberg] Carpenter, G. and Grossberg S. (1987) "ART2: Self-organization of stable category recognition codes for analog input patterns" Applied Optics, 26, 4919-4930

[Cooper *et al.* 88] Cooper, L.N., Reilly D.E. and Elbaum, C. (1988). *Neural Network Systems, An Introduction for Managers, Decision-Makers and Strategists* Nestor, Inc. One Richmond Square, Providence, Ri 02906 (401) 331-9640. 31 pages.

[Depenau and Møller 94] Depenau, J. and Møller, M. (1994), *Aspects of Generalization and Pruning*, Proceedings from the World Congress on Neural Networks, Vol 3, pp. 464-469, San Diego 1994.

[Depenau 95] Depenau, J. (1995), *A Global-Local Learning Algorithm*, Proceedings from the World Congress on Neural Networks, Vol I, pp. 587-590, Washington 1995.

[Depenau 94] Depenau, J. (1994), *Evaluation of the Cascade-Correlation Algorithm*, Technical Report.

[Fahlman and Lebiere 90] Fahlman, S.E. and C. Lebiere (1990), *The Cascade-Correlation Learning Architecture*. In Advances in Neural Information Processing Systems II (Denver 1989), ed. D.S. Touretzky, 524-532. San Mateo: Morgan Kaufmann.

[Fahlman 89] Fahlman, S.E. (1989), *Fast Learning Variations on Back-propagation: An Empirical Study*, In proceedings of the 1988 Connectionist Models Summer School, Eds. D.S. Touretzky, G. Hinton and T. Sejnowski, pp. 38-51, San Mateo: Morgan Kauffmann.

[Finnoff *et al.* 93] Finnoff, W., Hergert, F. and Zimmermann, H.G. (1993) *Improving Model Selection by Nonconvergent Methods*. Neural Networks. Vol. 6, pp. 771-783.

[Frean 90] Frean, M. (1990), *The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks*, Neural Computation, Vol. 2, pp. 198-209.

[Fritzke 94] Fritzke, B. (1994), *Growing Cell Structures - A Self-Organizing Network for Unsupervised and Supervised Learning*, Neural Networks, Vol. 7, pp. 1441-1460, 1994.

[Gallant 86] Gallant S.I. (1986), *Three Constructing algorithms for Network learning*, Proc. 8th Annual Conf. of Cognitive Science Society, 652-660.

[Gallant 86] Gallant S.I. (1986), *Optimal Linear Discriminants*, In Eighth International Conference on Pattern Recognition (Paris 1986), pp. 849-852. New York: IEEE.

[Gallant 90] Gallant, S. (1990), *Perceptron-Based Learning Algorithms*, IEEE Transactions on Neural Networks, Vol. 1, No. 2, pp. 179-191.

[Hansen and Salamon 92] Hansen, L.K. and Salamon, P. (1992), *Neural Network Ensembles*. IEEE Trans. Pattern Analysis and Machine Intell., pp. 993 - 1001.

[Harp *et al.* 90] Harp, S.A., Samad, T. and Guha, A.(1990), *Designing Application-Specific Neural Networks Using the Genetic Algorithms*. Advances in Neural Information Processing Systems II (Denver 1989). ed. D.S. Touretzky, pp. 447-454. San Mateo: Morgan Kaufmann.

[Hassibi and Stork 93] Hassibi, B., and Stork, D. (1993), *Second order derivatives for network pruning: Optimal Brain Surgeon*. Advances in Neural Information Processing Systems V (Denver 1993). ed. S.J. Hanson et al., 164-171. San Mateo: Morgan Kaufmann.

[Huang and Huang 91] Huang, S,C. and Huang Y.F. (1991), *Bounds on the Number of Hidden Neurons in Multilayer Perceptrons* IEEE Transactions on Neural Networks, Vol. 2, No. 1, pp. 47-55.

[Hwang *et al.* 94] Hwang, J. You, S., Lay. S. and Jou, I. (1994), *What's Wrong with A Cascade-Correlation Learning Network: A Projection Pursuit Learning Perspective*. Found on connectionist ftp list.

[Knudsen and Vlk 94] Knudsen, S.L. and Vlk, L.H. (1994), *Ockham's Razor Drawn against Overfitted Neural Nets: A Quest for the Glorious Features*, Masters Thesis, Daimi IR-121, Computer Science Department, Aarhus University.

[Le Cun *et al.* 90] Le Cun, Y., Denker, J.S. and Solla, S.A. (1990), *Optimal Brain Damage*. In Advances in Neural Information Processing Systems II (Denver 1989). ed. D.S. Touretzky, 598-605. San Mateo: Morgan Kaufmann.

[Marchand *et al.* 90] Marchand, M.M., Golea, M. and Ruján, P. (1990), *A Convergence Theorem for Sequential Learning in Two-Layer Perceptron*, Europhysics Letters 11, pp. 487-492, 1990.

[Mezard and Nadal 89] Mezard, M. and Nadal, J.P. (1989), *Learning in Feedforward Layered Networks: The Tiling Algorithm*, J. Phys. A: Math. Gen., Vol. 22, pp. 2191-2203.

[Miller *et al.* 89] Miller, G.F., Todd, P.M. and Hegde, S.U. (1989), *Designing Neural Networks Using Genetic Algorithms*. In Proceedings of the Third International Conference on Genetic Algorithms, ICGA'89. Arlington 1989), Ed. J.D. Shaffer, pp. 319-384. San Mateo: Morgan Kaufmann.

[Pedersen 95] Pedersen H.C.(1995), *Clustering*. Report to Neural Network Studygroup (in Danish), Computer Science Department, Aarhus University.

[Platt 91] Platt, J.C. (1991), *Learning by Combining memorization and Gradient Descent*, in NIPS 3, Editors D. Touretzky, M. Kaufmann Publishers, Inc., Denver, Colorado, pp. 714-720.

[Reed 93] Reed, R. (1993), *Pruning Algorithms - A Survey*, IEEE Transactions on Information Neural Networks, Vol. 4, No. 5, pp. 740-747.

[Reilly *et al.* 82] Reilly D.E., Cooper, L.N. and Elbaum, C. (1982). *A Neural Model for Category Learning. Biological Cybernetics*, Vol 45, pp. 35-41, Springer-Verlag Berlin-Heidelberg.

[Weigend et al. 90] Weigend, A.S., Huberman, B.A. and Rumelhart, D.E. (1990), *Predicting the Future: A Connectionist Approach*, International Journal of Neural Systems, Vol. 1, pp.193-209.

[Whitley *et al.* 90] Whitley D., Starkweather T. and Bogart C. (1990), *Genetic algorithms and neural networks: optimizing connections and connectivity.* Parallel Computing 14, 1990, pp 347-361. Elsevier Science Publishers B.V. (North-Holland).

[]**RELATED TOPICS**

[Dasarathy 90] Dasarathy, B. V. (1990), *Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press.

[Duda and Hart 73] Duda, R.O. and Hart, P.E. (1973), *Pattern Classification and Scene Analysis.* Wiley, New York.

[Fletcher 75] Fletcher, R. (1975), *Practical Methods of Optimization*, Vol. 1, John Wiley & Sons.

[Fukunaga 90] Fukunaga, K. (1990), *Introduction to Statistical Pattern Recognition.* ACADEMIC PRESS, New York.

[Gill *et al.* 81] Gill, P.E., Murray, W. and Wright, M.H. (1981), *Practical Optimization*, Academic Press.

[Goldberg 89] Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

[Golub and Loan 83] Golub, G.H. and van Loan, C.F. (1983), *Matrix Computations*, The John Hopkins University Press.

[Haralick 71] Haralick, R.M. (1971,) *On a Texture-Context Feature Extraction Algorithm for Remotely Sensed Imagery*, IEEE Decision and Control Conference, pp. 650-657.

[Haralick 79] Haralick, R.M. (1979), *Statistical and Structural Approaches to Texture*, Proc. IEEE 67, pp. 786-804.

[Haralick *et al.* 73] Haralick, R.M., Shanmugan, K.S. and Dinstein, I. (1973), *Textural Feature for Image Classification*, IEEE Trans. Syst., Man, Cybern., SMC-3, pp. 610-621.

[Holland 75] Holland, J.H. (1975), *Adaption in Natural and Artificial Systems.* University of Michigan Press.

[Horn and Johnson 85] Horn, R.H. and Johnson, C.A. (1985), *Matrix Analysis*, Cambridge University Press, Cambridge.

[Knuth 81] Knuth, D.E. (1981), *The Art of Computer Programming*, Vol. 2, Semi-Numerical Algorithms, Addison-Wesley Publishing Company.

[Kreyszig 88] Kreyszig, E. (1988), *Advanced Engineering Mathematics*, 6th edition, John Wiley and Sons, Inc.

[Leamer 79] Leamer E.E (1979), *Specification searcher*. New York: John Wiley and Sons.

[Luenberger 84] Luenberger, D.G. (1984), *Linear and Nonlinear Programming*, Addison-Wesley Publishing Company, Inc.

[Niblack 85] Niblack, W. (1985), *An Introduction to Digital Image Processing*, Strandsberg Publishing Company, Birkerød, Denmark 1985.

[Skriver 95] Skriver, H. (1995), *Private communication*.

[Skriver 94] Skriver, H. (1994) *On the accuracy of estimation of ice type concentration from ERS-1 SAR images* from EARSEL 14th Symposium, Gothenburg, 1994).

[Skriver 89] Skriver, H. (1989), *Extraction of Sea Ice Parameters from Synthetic Aperture Radar Images*, Ph.D. Thesis from Electromagnetic Institute, Technical University of Denmark, Lyngby, 1989.

Appendix A

Glossary

This glossary gives a brief explanation of terms used in this thesis.

Activity: The value of a unit calculated by a combination between the input to the unit and its incoming weights.

Architecture: The architecture of a neural network determines the number of neurons in the network and the topology of the connections within the network.

Back-Propagation: A supervised training algorithm for neural networks which makes use of the difference between the calculation made by the network and the desired output set by the experimenter. In order to adjust the weights in the network, the error is assigned to the units in proportion to their contribution to the final output. This assignment begins with the output layer and works backwards down the network through the hidden layer(s) towards the input layer.

Bayes Optimal: The Bayesian classifier is theoretically the best classifier assuming that the distribution of patterns is given. It has shown to be optimal in the sense that it minimises the probability of classification error. Bayes optimal refers to the property that the use of a particular error function in connection with a learning method should be able to adjust the parameter of a model/network so that it approaches the lower bound to error (the Bayesian classifier bound) as the number of training data approaches infinity.

Binary function: A function which transfers a number of inputs (real value) into one of two possible states, often -1 and 1.

Capacity: An expression for how many different functions a network with a given architecture can implement. It is often expressed by the $\mathcal{VC}_{dim}$.

Cluster: A number of mutually exclusive classes containing patterns in the same class that are similar to one another according to some measurement of similarity, while patterns in different classes are dissimilar.

Complexity: The complexity of a feed-forward neural network is related to the number and topology of units and weights. In general one could say that a multi-layer perceptron is more complex (has a higher degree of complexity) than a simple perceptron.

Conjugate Gradient: One of several supervised training algorithms for making a multi-layer feed-forward network learn faster. Basically a minimisation technique which uses multi-dimensional gradient information to determine in which direction(s) to search, moving in each direction without undoing the effect of previous directions. The gradient information is calculated using algorithms such as error Back-Propagation.

Confusion Matrix: A graphical way (table/matrix) to evaluate the performance of a classifier. The rows in the confusion matrix correspond to the true classes of the pattern presented to the network (the desired class) and the columns correspond to the output of the network. An entry represents the frequency with which a network makes a certain classification. As an example, the fourth entry in the second column is the number of times that the network has classified a pattern from class 2 as belonging to class 4. A good confusion matrix would be one with large numbers in the diagonal and small elsewhere.

Construction algorithms: A general expression for methods that successively build neural networks.

Covariance Matrix: The covariance of two variables X and Y is a measure of the relation between the two variables. The covariance matrix denoted $Cov[X, Y]$, is defined as:

$$Cov[X, Y] = \frac{1}{P} \sum_{i}^{P} \sum_{i}^{P} (X_i - \overline{X})(Y_j - \overline{Y})$$

where $\overline{X}$ denotes the mean of X, $\overline{Y}$ denotes the mean of Y, and P the number of samples (data). The covariance matrix is a square matrix whose (i, j)'th entry is the covariance of the i'th and j'th variables.

Cross Validation: A technique for estimating the generalisation error by an error computed empirically on the basis of data available in the form of a data set. The basic idea is to split the data set into two, one part of data for training (and training only) and another used for test (and test only), so $P = P_{Training} + P_{Test}$. The generalisation error is simply approximated as:

$$E_G(\mathbf{W}) \approx \frac{1}{P} \sum_{p=1}^{P_{Test}} E(\mathbf{V}_L^p, \mathbf{T}^p)$$

Early Stopping: A learning strategy where the training is stopped before the network has made use of many of its degrees of freedom and over-fitting appears. The data set is split into three sets, a training set, a test set and a validation set. The training set is used in the usual way while the validation set is used for test during learning. The Early Stopping is performed by observing the error on the validation set and then stopping the training as soon as the error starts to increase.

Empirical Risk Minimisation (ERM): The ERM theory builds on the idea of uniform convergence of empirical risk to actual risk. In other connections the Empirical Risk is

called the training error and the actual risk is called the generalisation error. The uniform convergence property describes their mutual relation and is expressed in the following way:

$$P(\sup_{\mathbf{w}\in\mathbf{W}} |E_G(\mathbf{W}) - E_L(\mathbf{W})| > \epsilon) \longrightarrow 0 \quad \text{as} \quad p \longrightarrow \infty$$

where "sup" denotes supremum, $\mathbf{W}$ is the space of all possible weight combinations in a network (also called the weight space), and ϵ is the error.

Error function: A function used to measure the error or difference between the output of the network and the desired output. It is often also called Cost-function, Energy function and Lyapunov function.

Error propagation: A method of calculating the partial derivative of a network's error with respect to each of its weights. The calculation starts by calculating the error at the network's output, and works back towards the input, hence the name. The technique is used in conjunction with an algorithm that makes use of the gradient information, such as Back-Propagation or Conjugate Gradient.

Feature: A particular characteristic of the data, such as statistical moments.

Generalisation ability: The expected performance on unknown data also commonly used as an expression of the ability of the model/machine/network to capture the underlying truth. It is an opposite equivalent to the generalisation error.

Generalisation error: The expected error on unknown data is called the generalisation error. It is defined as an average over the full distribution of input-output pairs and can be expressed as:

$$E_G(\mathbf{W}) = \int E(\mathbf{V}_L, \mathbf{T}) P(\mathbf{V}_0, \mathbf{T})\, d\mathbf{V}_0\, d\mathbf{T}.$$

where $P(\mathbf{V}_0, \mathbf{T})$ is the joint probability distribution formed by $P(\mathbf{V}_0)$ describing the distribution of data in the input space and $P(\mathbf{T}|\mathbf{V}_0)$ describing the functional dependence, so $P(\mathbf{V}_0, \mathbf{T}) = P(\mathbf{T}|\mathbf{V}_0)P(\mathbf{V}_0)$. The generalisation error is the expectation value of error for an arbitrary $(\mathbf{V}_0, \mathbf{T})$ point drawn from the $P(\mathbf{V}_0, \mathbf{T})$ distribution.

Genetic Algorithms (GAs): Techniques for optimisation inspired by the process of natural evolution in population biology. A genetic algorithm maintains a *population* of *individuals* each of which corresponds to a specific solution to the given optimisation problem. A measure of *fitness* defines the quality of a solution. Starting from a population consisting of randomly generated individuals, the evolution process is simulated by considering the population through a number of *generations*. In each generation, new individuals called *offspring*, are generated from existing ones using a *crossover* operator, which imitates sexual propagation. The crossover operator is designed in such a way that the generated offspring resembles the parent individuals. Furthermore, parents are selected for crossover with a probability which depends on their fitness, so that the fittest individuals are selected for crossover with the highest frequency. This scheme enforces the principle of survival of the fittest. With a small probability, each individual is subjected to mutation, or random change, by the *mutation* operator. After having simulated a number of generations, highly fit individuals will emerge, corresponding to good solutions to the

given optimisation problem. A distinction is made between the representation, or genetic encoding of a solution and the natural appearance of the solution. By analogy with biology, the genetic encoding is called the *genotype* and the natural appearance is called the *phenotype*. The genetic operators manipulate solutions in terms of their genotypes, while fitness is measured in terms of phenotype. A function called the *decoder* computes the phenotype corresponding to a given genotype. GAs have in many cases proved to be effective search strategies that can be applied to several different types of system. In connection with neural networks they have been used both for training the network and building the network.

Growth function: The growth function $\triangle(P)$ is defined as the maximum number of different binary functions that can be implemented by the network on a data set containing P patterns.

Indicator function: The function used in connection with classification as an error function giving 1 if the classification is correct otherwise 0. In neural networks terms this means that $E(\mathbf{V}_L, \mathbf{T}) = 0$ if $f_W(\mathbf{V}_0) = \mathbf{T}$, and $E(\mathbf{V}_L, \mathbf{T}) = 1$ otherwise.

K-mean clustering: An algorithm that partitions the patterns into k classes, so that the total sum of distance (or similarity) within classes is minimised.

Layer: In several neural network models the units are organised in layers. The input pattern is presented to the network via the input layer, often just a set of data latches, but sigmoid functions may also be applied to normalise the input. Processing then passes through one or more layers of hidden units, operating in parallel, before reaching the output layer. The number of units in the input and output layers is generally dictated by the input data and the type of response required from the network. The nature of the data and the success of the training process are factors that affect the number of units needed in the hidden layer and, indeed, the number of hidden layers required.

Learning machine: A learning machine, broadly defined, is any device whose actions are influenced by past experiences. This thesis deals with a subclass of learning machines, those which can be trained to classify patterns.

Leave-one-out and N-fold Cross Validation: For an N-fold cross validation, the data are divided into N equal and disjoint sets. Each of these sets is left out in turn, and the network is trained on data from the remaining $N-1$ sets. It is then tested on the set which was left out. This procedure is repeated until all N sets have been used as test set and the final test set error is calculated as the average of all test errors. A very popular version of this method is to use the test sets with only single data, the method is called Leave-one-out.

Linear transfer function: A function in which the output is equal to a gain value times a weighted sum of the inputs.

Linearly separable: If a hyper-planar decision boundary exists that correctly classifies all training data as belonging to one of two classes, the training data are linearly separable.

Line search: A method used in connection with optimisation to find the best step-size (also called learning-rate). In many optimisation techniques the gradient or a combination of gradients is used to determine the direction in which the parameters (weights) changes. In order to find how much the parameters should change, a search starting at the current parameter position in the direction of the gradient (or combination of gradients) is performed to find the minimum of the function. In the two-dimensional case the starting point and the gradient describe a line so the search of the functions minimum is done along a line.

Multi-layer feed-forward network: Standard neural model accounting for probably over 90% of current applications. Usually trained by Back-Propagation.

Neural Network: Conceptually a neural network consists of a number of relatively simple processing elements (units) that are interconnected. Each unit element receives a number of input signals, multiplies each by its associated "weight", and sums the results. This total is put through a non-linear transformation and the result is passed on to later processing stages.

Neuron: A single processing element within a neural network. In this thesis it is called a unit.

Ockham's Razor: William of Ockham (c. 1280 - 1349) is considered the most influential philosopher of the fourteenth century. He frequently employed a principle of methodological economy in explanation, which is known as *Ockham's Razor*. He used it in such forms as "Plurality is not to be assumed without necessity" and "What can be done with fewer is done in vain with more". It is popular to interpret the latter in connection with neural networks as:

> *Among several neural network models that are able to learn the training data in a satisfactory way, the one using the smallest number of parameters is the network with the largest probability of a good generalisation ability.*

Off-line learning: A training method where all the training data are presented to the network before the weights are updated.

On-line learning: A training method which enables the network to learn as the training data becomes available. In the learning phase the weights are updated every time a pattern is presented to them.

PAC (Probably Approximately Correct Learning Theory): A theory based on the intention that successful learning of a function f^d (^{d} = desired) taken from a class of functions $\mathcal{F}$ should, with high **probability**, be learnt by a function $f_{\mathbf{w}}$ (network) so that it is a good **approximation** of f^d.

Perceptron learning rule: An algorithm for training a simple perceptron with threshold transfer functions. It is based on the idea of changing the weights in the following way:

$$\Delta w_{ij} = \eta\Theta(N\kappa - T^p i U_i^p)T^p i V_j^p$$

where η is the learning rate, N the number of inputs, κ is a margin, p is the current pattern number, and $\Theta(x)$ is a function which is 1 if $x > 0$ and 0 otherwise. The weights correspond to the parameter of the hyper-plan which the threshold function implements. This equation says that the weights should be changed in the direction of a pattern whose output agrees with the target. By showing the patterns to the perceptron over and over again the perceptron learning rule will eventually (in a finite number of steps) come up with a hyper-plan that separates all patterns correctly if such a solution exists (i.e. if the training data are linearly separable).

Pocket algorithm: An algorithm for training a simple perceptron with threshold transfer functions. It is an extension of the perceptron learning rule. An unmodified perceptron learning rule wanders through weight space when the problem is not linearly separable, spending most time in regions giving the fewest errors, but not staying there. So the pocket algorithm modification consists simply in storing (or "putting in your pocket") the set of weights which has had the longest unmodified run of successes so far. The algorithm is stopped after a chosen time t.

Pruning: A common expression for methods used to reduce the complexity i.e. number of weights and sometimes units in a well working feed-forward network without changing the learning error significantly.

Radial Basis Function Network: The method of Radial Basis Functions (RBFs) is a powerful neural network technique that is derived from conventional approximation theory. An RBF network consists of a single hidden layer of units whose output depends on a distance between the centre of an arbitrary transfer function and a given pattern. The output layer will commonly consist of standard linearly weighted units with a linear activation function.

Regularisation: A method based on the idea that the generalisation error is a sum of two components: the error on the training set and an additional term depending on the network's complexity. The generalisation error $E_G(\mathbf{W})$ is approximated in the following way:

$$E_G(\mathbf{W}) \approx \sum_{p=1}^{P} E(\mathbf{V}_L^p, \mathbf{T}^p) + \lambda C(f_{\mathbf{w}})$$

where $C(f_{\mathbf{w}})$ is a monotonic increasing cost-function of the mapping function $f_{\mathbf{w}}$ as the complexity of $f_{\mathbf{w}}$ increases. α is the regularisation parameter that controls the compromise between the degree of smoothness of solution and its closeness to the data.

Saturation: The state of a unit using sigmoid or threshold transfer function when a small change in the activity value does not change the output value.

Self-Organising Map (SOM): Un-supervised trained network which is used to cluster input signals into groupings according to their similarity. The network is often called Kohonen Self-Organising Map or Kohonen network after its inventor. The network consists typically of an input layer and an output layer arranged in a 2-dimensional array, or "map", also known as a "feature map". The output units are not only connected to the input but also to each other. The weight update between input to output units is done

so that the output unit with the strongest response (the winner) becomes stronger while the response from the rest becomes smaller (this is normally called competitive learning). The weights between the winning output unit and output units that are located close to it are then updated, typically so that the "cluster" around the winner gives a stronger response.

Shuffle: A technique used to escape local minima by ensuring that data from the same class are not presented one after the other all the time and that the order of the data is changed from epoch to epoch.

Supervised learning: Training process in which each data pattern is assigned to a particular class by the experimenter. During training the network is told, for each pattern presented, which class it belongs to, and thus learns to associate relevant patterns with each class. During recall, each input pattern generates an output indicating the appropriate class.

Taylor approximation: A way to approximate a function by a power series where the coefficients are determined by higher and higher degrees of the derivatives. Let $f(x)$ be a differentiable function. The Taylor Expansion says that the value of $f(x_1)$ can be calculated on the basis of the current value of the function $f(x_0)$ and a number of derivatives in the following way:

$$f(x_1) \approx f(x_0) + \frac{1}{1!}f'(x_0)(x_1 - x_0) + \frac{1}{2!}f''(x_0)(x_1 - x_0)^2 + \cdots + \frac{1}{N!}f^{(N)}(x_0)(x_1 - x_0)^N$$

Since the Taylor Expansion (of degree N about x_0) uses a finite number of terms it is only an approximation of the real function and therefore also called Taylor approximation.

Test Error: In connection with Cross Validation the data set is often split into two, one part of data for training (training set) and another only used for test (test set), so $P = P_{Training} + P_{Test}$. The error on the test set called the test error is:

$$E_{Test}(\mathbf{W}) = \frac{1}{P}\sum_{p=1}^{P_{Test}} E(\mathbf{V}_L^p, \mathbf{T}^p)$$

It is common to use the test error as an approximation of the generalisation error.

Training Algorithms: For each neural network model, there is an associated range of algorithms which can be used to establish the desired input-output function to be implemented. In general, these training algorithms are based on techniques for mathematical optimisation, such as the principle of Gradient Descent. Each of the particular training algorithms has its own characteristics, which make it suitable for different applications. The most well-known training algorithm is termed 'Back-Propagation of errors' and is used to train a multi-layer feed-forward network.

Transfer function: Function applied to calculate the output of a unit (a function that transfer the activity of a unit to its output). It is often non-linear, e.g. sigmoid or Gaussian, in order to enable the neural network to perform general function approximation.

Unit: A single processing element within a neural network.

Un-supervised learning: The network is presented with examples of the data, and finds the major differentiating features by itself, e.g. Kohonen's self-organising map (SOM).

VC-dimension: The VC-dimension denoted $\mathcal{VC}_{dim}$ describes the maximum number of unlabelled patterns for which all possible binary labellings can be learned without error.

Weight: A value, held for each conceptual connection in a neural network, which represents the importance of the signals down that connection when computing the response of the network.

Weight Decay: Methods based on the regularisation technique where the additional term tries to penalise weights with large values. The simplest way of weight decay is to let the additional term be a sum of square weight values, so the error function becomes:

$$E_G(\mathbf{W}) \approx \sum_{p=1}^{P} E(\mathbf{V}_L^p, \mathbf{T}^p) + \frac{1}{2}\lambda \sum_{l=0}^{L} \sum_{k=0}^{N_l} w_{lk}^2$$

where l and k are indexes to all weights in the network.

Appendix B

Educational Plan for EF-448

This Appendix contains the Educational Plan which provides the background for the Industrial Research Education No. 448.

B.0 History

At the start of the project the original task was expected to be classification of formatted messages and the project was named: Formatted Message Analysis by Neural Networks. It turned out that the data would not be accessible within the period of this project. The Educational Plan was therefore modified in November 1994 so it would be up to date with the new problem. The updated version was, however, written as if it was done at the start of the project, in order to match the original plan as much as possible, and thereby securing that there would be no problem having it accepted by the Danish Academy of Technical Sciences (ATV). It was accepted by ATV in March 1995.

Appendix C

Aspects of Generalization and Pruning

This paper was handed out and presented at the *Ph.D. Summer School in Data Complexity*, DIKU, Copenhagen, 7 - 13 August 1994.

A shorter version was presented at a poster session at the *WCNN'94 (World Congress on Neural Networks),* San Diego, Califonia, 3 - 10 June 1994 and can be found in the proceedings from this Congress, see:

Aspects of Generalization and Pruning, Proceedings from the World Congress on Neural Networks, Vol 3, pp. 464-469, San Diego 1994. J. Depenau and M. Møller.

Appendix D

Soft-Monotonic Error Functions

This paper was presented at an oral session at the *WCNN'94 (World Congress on Neural Networks),* San Diego, Califonia, 3 - 10 June 1994 and can be found in the proceedings from this Congress, see:

Soft-Monotonic Error Functions, Proceedings from the World Congress on Neural Networks, Vol 3, pp. 444-449, San Diego 1994. M. Møller and J. Depenau.

Appendix E

Evaluation of the Cascade-Correlation Algorithm

This paper was published as a technical report and sent to colleagues and fellow researchers in late October 1994.

E.0 Remarks

The report suggested several new ideas. After the release of the report the following month was spent working on this and other ideas in collaboration with Morten Buhr and Martin Møller, but without the great success we hoped for. The problem was in general that there was no guarantee that the units inserted would do anything useful. We ended up concluding that the new ideas did not represent the road to follow and therefore this report was never updated as intended. Instead the search for a new method that would guarantee that the inserted unit would at least improve the learning error, lead to the developement of GLOCAL described in chapter 6.

Appendix F

A Global-Local Learning Algorithm

This paper was presented at an oral session at the *WCNN'95 (World Congress on Neural Networks),* Washington D.C., 17 - 21 July 1995 and can be found in the proceedings from this Congress, see:

A Global-Local Learning Algorithm, Proceedings from the World Congress on Neural Networks, Vol 1, pp. 587-590, Washington 1995. J. Depenau.

Appendix G

The MS-σ Error Function

This paper was presented at an oral session at the *WCNN'95 (World Congress on Neural Networks),* Washington D.C., 17 - 21 July 1995 and can be found in the proceedings from this Congress, see:

The MS-σ Error Function, Proceedings from the World Congress on Neural Networks, Vol 1, pp. 614-617, Washington 1995. J. Depenau and M. Møller.