Contents

List of Tables

List of Figures

Chapter 1

Prelude

In this thesis the two disciplines of Statistics and Artificial Neural Networks are combined into an integrated study of a data set of particular interest. These two methodologies have existed side by side for many years and only in recent times have the paths of the proponents of the two disciplines crossed.

Statistics is one of the oldest disciplines concerned with studying data and making inferences based on the information contained in the data - its origins can be traced back to the middle of the seventeenth century. Statistical modelling is the process of fitting statistical models to data from practical problems, testing the adequacy of these models, and finally extracting useful information from them.

Artificial neural networks, on the other hand, owes its existence to modern technology. The first artificial neural networks were designed as recently as sixty years ago. In essence, artificial neural networks are mathematical models for information processing which employ algorithms to approximate real-valued, discrete-valued or vector-valued target functions using a set of input-output pairs. Without powerful computers to execute the often computationally intensive algorithms, these systems would never have been developed and refined to such an extent that the field is now strongly interdisciplinary, in-

volving computer scientists, mathematicians, meteorologists, engineers, physicists, psychologists and financial analysts to name but a few.

Historically, the major developments in artificial neural network methodology have progressed for the most part independently of the discipline of Statistics. This has inevitably resulted in the duplication or reinvention of procedures as the two fields are in many respects closely related: both developed methodologies with the aim to learn, or predict, from examples.

Modern technology has managed to remove the invisible and intangible boundaries between the two disciplines. Huge amounts of information can now be exchanged and accessed almost instantaneously via the internet. As artificial neural networks are becoming increasingly popular with data analysts - who do not necessarily have a statistical background - via readily available and easily implemented software packages, more statisticians are realizing the need to become involved in the field. It is now widely recognized that statisticians can make valuable contributions to artificial neural network methodology.

One of the objectives of this thesis is to bring artificial neural network methodology into the home of statisticians. Chapters 2 to 4, comprising the literature study, were written with this goal in mind.

Another objective is to demonstrate the important role that statisticians can play in further developing and refining neural network methodology by applying existing statistical techniques in a novel way. A pruning algorithm that was developed to optimize the number of hidden nodes in a feedforward artificial neural network trained by backpropagation illustrates this aim. This recursive algorithm improves upon the way in which the artificial neural network architecture is determined, and is described in detail in Chapter 5.

A nonlinear regression statistical model setting was used to construct the algorithm. In the algorithm the results of one completed training session of an artificial neural network is statistically analysed to determine and specify the

number of hidden nodes that can be eliminated from a neural network model under construction. The procedure is based on a singular value decomposition of the conditional information matrix. It uses likelihood-ratio test statistics as selection criteria for the specific nodes to be eliminated, as well as for the selection of the correct artificial neural network model. Implementation of the algorithm dramatically reduces the number of training sessions necessary to find a model that fits the data adequately. This original contribution is especially valuable to artificial neural network users who do not use neural network packages, but program the instructions instead, as reported at the IEEE World Congress on Computational Intelligence [Fletcher & Engelbrecht, 1998].

The third objective of this thesis was to statistically model weather modification data using both an artificial neural network and a classical statistical technique to ascertain how well an artificial neural network model performs on a smaller data set in comparison with an analogous statistical technique.

The need to apply an artificial neural network to this type of data was prompted by the increasing implementation of artificial neural networks in the field of Meteorology. A quick glance at the literature shows numerous applications, e.g. [Jones *et al.*, 1999], [Narasimhan *et al.*, 2000], [Liu *et al.*, 2001] and [Silverman & Dracup, 2000], all of them using large data sets. In practical applications artificial neural networks typically deal with very large data sets where its effectiveness as a modelling tool has been widely proved. The effectiveness of the application of artificial neural networks to smaller data sets has, however, as far as could be established, not been well researched or well documented.

Chapter 6 contains a description of how a discriminant analysis and a multilayer perceptron artificial neural network were applied to a relatively small data set. The formulation of the problem, the approach that has been followed to solve it and the novel modelling application all combine to make

an original contribution to the interdisciplinary fields of Statistics and Artificial Neural Networks as well as to the discipline of Meteorology. The results have been presented at two conferences in 2002: [Fletcher & Steffens, 2002a] and [Steffens & Fletcher, 2002]. A paper has been accepted to be read at the 2002 Conference of the South African Society for Atmospheric Sciences [Fletcher & Steffens, 2002b]: this is the forum where the results will be formally communicated to the meteorologists.

The thesis is set out as follows:

In Chapter 2, statistical modelling and predictive learning are briefly introduced. An artificial neural network is described in the context of its neurological counterpart and the differences and similarities between the two approaches highlighted.

Chapter 3 gives an overview of the historical development of artificial neural networks, starting with the pioneering publication "A Logical Calculus of the Ideas Immanent in Nervous Activity" by Warren McCullogh and Walter Pitts in 1943 [McCulloch & Pitts, 1943]. Some of the most important contributions over the past decades that had a significant impact on the evolution of the field of artificial neural networks are discussed in the subsequent sections.

The various aspects concerning the design and training of an artificial neural network are discussed in Chapter 4. The aim in Chapter 4 was to extract and assimilate from the vast body of literature on artificial neural networks a coherent whole that sets out and explains the various components of artificial neural network systems in a logical way.

Quality data form a crucial part in the development of an artificial neural network, therefore, the collection, auditing and preprocessing of the data are discussed first. This section was motivated by the erroneous and uninformed view often held by users that artificial neural networks will, by virtue of some "blackbox" magicking, extract the necessary and useful features from any data

set regardless of its nature. In the next section the architecture of the network in single or multiple layers is explained. Thereafter, commonly used artificial neural network activation functions are presented.

Following that, a number of different artificial neural network learning rules which are widely employed are introduced. Hebb's learning rule, Frank Rosenblatt's perceptron learning rule, the ADALINE, the delta learning rule, Kohonen self-organizing maps and adaptive resonance theory are outlined. This is done under the three broad classes of artificial neural network models that can be identified based on the type of training, namely self-supervised, supervised and unsupervised.

The next section explains the simple gradient descent method to minimize the total squared error of the output computed by the artificial neural network, called the backpropagation of error, in some detail. Backpropagation is the most popular algorithm for adjusting weights during the training phase of a feedforward artificial neural network, to the extent that the three layer backpropagation network (i.e. an input layer, one hidden layer and an output layer) has become the industry standard.

Chapter 4 concludes with a short discussion of validation, multiple random starts and recutting as suitable techniques for evaluating the performance of a neural model.

In Chapter 5, the original recursive algorithm that was developed to optimize the number of hidden nodes in a feedforward artificial neural network is described in detail. The chapter starts with the setting of the problem and with an explanation of the statistical theory and method. Thereafter the two stages of the optimization algorithm is described. The chapter concludes with a simulation study to illustrate the effectiveness of the algorithm.

The third objective of this thesis, i.e. the statistically modelling of weather modification data, is addressed in Chapter 6.

In this chapter, the problem with selecting appropriate storms for seeding in weather modification experiments that has been identified whilst analyzing the results of the National Precipitation Research Programme conducted in South Africa during the early 1990s is described. Previous analyses indicated that the mean and median radar-measured rain masses of seeded storms were significantly higher than those of unseeded storms [Fletcher & Steffens, 1996] and [Mather & Fletcher, 1997]. Studies of the time histories of individual storms, however, highlighted the phenomenon that not all seeded storms had responded positively to seeding as some of these storms produced very little rain after seeding. The challenge was to be able to correctly identify appropriate storms for seeding before the seeding decision is taken as this will aid in better selection of storms.

Discriminant analyses and a multilayer perceptron neural network were used in this chapter to develop models that classify storms into two groups: those which seemingly behaved like seeded storms and those which did not, based on their rain mass evolution over time. Thereafter, oneway analyses of variance were performed to compare these two groups with respect to the means of several explanatory variables representing different storm properties such as echo tops, storm depth, storm volume, storm mass, storm area, rain flux, precipitable water content and reflectivity, all based on radar measurements taken in the ten minutes before the seeding decision was taken. The aim was to obtain an indication of possible variables that may be useful in distinguishing between storms which seemed to have reacted positively to seeding and those which did not.

The thesis concludes with a summary of its contents in Chapter 7 and with a brief outline of proposals for further research initiatives emanating directly from the research conducted as well as conceived by the stimulation of the research process.

Chapter 2

Statistical modelling and artificial neural networks

Statistical modelling and artificial neural networks are briefly introduced in this chapter. An artificial neural network is described in context with its neurological counterpart and the differences and similarities between the two disciplines are summarized.

2.1 Statistical modelling

Statistics is one of the oldest disciplines to study data and make inferences based on the information contained in the data. Statistical modelling is required whenever such information has to be gleaned from data. Statistical modelling comprises the fitting of statistical models to data from practical problems, the testing of the adequacy of these models, and finally the extraction of useful summary information from them.

In practice, statisticians rely on constructing models of "causal situations" in order to explain and predict satisfactorily what is happening, and hence to draw valid inferential conclusions from the data. Statistical models, as in all

areas of science, thus serve as a portrayal of the reality. Hence their quality and usefulness is heavily dependent on the complexity of the model itself: simple models represent simple phenomena.

The numerical developments in the last decades, together with increasing advances in computing technology, have enabled researchers to analyse, investigate and explore large and complex data situations using innovative methods to build sophisticated statistical models. Much of the recent theoretical and technical developments in statistical modelling have been stimulated by data coming from various areas of science and industry and statistical modelling is now generally regarded as a multidisciplinary science.

In the last few years, two major branches of statistical modelling have evolved to accommodate the increasing demand for complex models portraying complex situations. These are mixed models and nonparametric models.

Mixed models allow the incorporation of unobserved heterogeneity and individual, latent effects into the model structure. It is possible, for example, to accommodate the overdispersion often found among outcomes that have nominally binomial or Poisson distributions, or to model the dependence among outcome variables inherent in longitudinal or repeated measures designs ([Breslow & Clayton, 1993], [Aitkin, 1999]).

Classical quantitative statistical models are restrained by being parametric, i.e. by attempting to portray the reality by a finite number of parameters. In nonparametric modelling, parametric functions are substituted by nonparametric flexible curves, allowing researchers to mirror and explore complex and totally unknown structures. Various different approaches have been followed in the last couple of years, some evolving from mathematics such as spline fitting, e.g. [Green & Silverman, 1994], others from extending statistical concepts such as Bayesian smoothing and local fitting, e.g. [Fan & Gijbels, 1995].

Model diagnostics has in the process emerged as an increasingly popular field of research. Generally speaking, every model is plausible as long as it is not "falsified" or misrepresented. The assessment, verification and comparison of different models is therefore of fundamental importance in order to be able to specify valid statistical models that fulfil the task of modelling reality. Both the theoretical aspects and applicability of the various routines used in the modelling process need to be evaluated in depth in order to understand and describe their impact. In the framework of nonparametric models, active research in the field of model diagnostics has led to several results, e.g. [Kauermann & Tutz, 1999]. For mixed models however, tools and concepts for model diagnostics are still rudimentary with resampling approaches appearing as the most popular line of research. The successful development of appropriate model diagnostic tools will enable researchers to draw valid inferential conclusions from their data, thus allowing scientists in all fields to gain from statistical modelling and to refine or correct their models.

Various workshops are annually dedicated to the dynamic field of research of statistical modelling, e.g. the Euroworkshop on Statistical Modelling, held from $1-4$ November 2001 in Bernried near Munich, Germany, focused on three themes, i.e. mixed models, nonparametric models and model diagnostics.

2.2 Predictive learning

Predictive learning systems attempt to construct accurate prediction rules using learning algorithms (generic computer programs) purely by processing the data without any domain specific knowledge. All information is presumed to be contained in the supplied data, and it is the function of the learning algorithm to automatically extract and organize that information to obtain the prediction rule.

In essence, Statistics has always been concerned with predictive learning. The methodology and theory of computer learning have traditionally been developed in the fields of Statistics (multiple regression and classification), Applied Mathematics (multivariate function approximation) and Engineering (pattern recognition).

The discovery of successful extensions to artificial neural networks during the 1980s, together with advances in computing technology, have led to renewed interest in computer learning and have generated research in both machine learning (artificial intelligence) and biologically motivated methods for data modelling (artificial neural networks). Unfortunately the major developments in these fields have progressed for the most part independently of each other with little cross-referencing in the literature, resulting in the reinvention of results in one discipline already well-known in other disciplines.

Statistics has possibly seen the greatest duplication of its procedures in other fields, probably because statisticians have been reluctant to adopt modern computer-based approaches. The cautious attitude of statisticians to data analyses that rely heavily on computer-based methods stems partly from the fact that it has been regarded the job of the scientist - not the statistician - to construct the structural model for the data. The role of the statistician is to analyse the data and study the inferential limitations of the scientist's model under various uncertainty conditions, i.e. to assess to what extent a collection of measurements actually characterizes the system as opposed to simply being an artefact of that particular sample.

Furthermore, statisticians have in the past been working mainly with data from relatively small samples with high noise to signal levels, mainly from fields such as medicine, psychology and political sciences. In such cases inference must be done with circumspection, using methods that have been validated mathematically. However, large data bases are now routinely generated by

systems for which the signal to noise ratio is high, especially in the engineering and physical sciences. Traditional statistical tools are invariably not flexible enough to extract all the available information from such data sets. It is the challenges associated with these types of data that are the main motivators for computer-based approaches (such as artificial neural networks) to predictive learning methods.

2.3 Artificial neural networks

Artificial neural networks evolved from the research objective to understand how the brain imparts abilities such as perceptual interpretation, associative recall (memory) and learning to humans, based on the neurobiological doctrine that the nervous system of living organisms is a structure consisting of many elements working in parallel with one another. The brain metaphor suggests that the brain's decision capability may be emulated by modelling the physical architecture of the brain within knowledge and capacity constraints.

The term neural network is hence derived from its biological similarity with the human brain, which is composed of neurons, each of which is connected to many others in a network that adapts and changes as the brain learns. The neuron cell of the brain was discovered in 1836 and its structure as a many-inputs/one-output unit earned its inventors, Camillo Golgi and Santiago Ramon y Cajal, the Nobel Prize in Physiology in 1906 (cf. http://www.nobel.se/medicine/laureates/1906).

Various features to increase the computational power of artificial neural networks have been included over the years, even though those features are not neurobiologically possible. These models, though inspired by brain function, only bear a metaphorical resemblance to natural biological neural networks.

In essence, artificial neural networks are mathematical models for information processing which employ algorithms to approximate real-valued, discrete-valued or vector target functions using a set of input-output pairs. The aim is to build models of data by capturing their most salient features during the training period (cf. Chapter 4). Artificial neural networks can be envisaged as functional approximators that fit the input and the output data with a high-dimensional surface.

A typical biological neuron receives input (either excitation or inhibition) from other neurons. The neuron fires when its net excitation reaches a certain threshold. The firing is propagated through a branching axon to many other neurons, where it in turn acts as input to those neurons. (The fact that the mind resides in the brain, which is packed with neurons, was widely accepted by 1930. Also known by then was the general nature of synapses and the threshold response of neurons.)

Figure 2-1 (taken from [Galkin, 2001]) illustrates a biological neuron with its axon and dendrites. The axons and dendrites carry the signals between neurons: axons allow a neuron to send a signal, whereas dendrites allow the neuron to receive signals from other neurons. The soma (cell body) takes cognisance of the incoming signals from the dendrites. The synaptic gaps are the junction parts of the neuron where the input signals are attenuated.

Hebb's learning rule states that a metabolic change occurs in the synapse when the input of a neuron is repeatedly and persistently causing the neuron to fire, reducing the synapse's resistance [Hebb, 1949]. The gaps thus change size in response to "learning" and are the regions where one cell excites or inhibits another cell. The cell is activated to fire or transmit a signal over its axon to other cells when sufficient input (synapse strength) is received, i.e. when some threshold is reached.

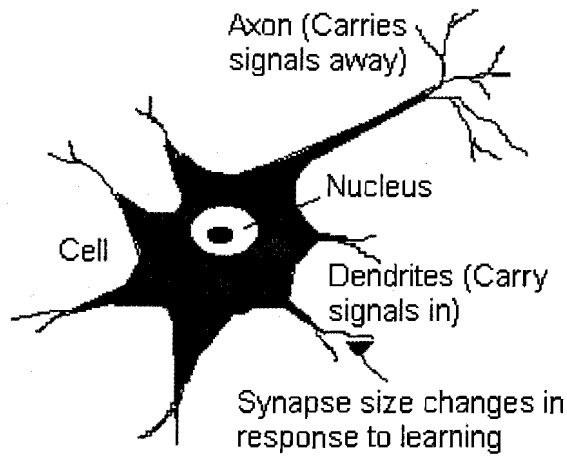

Figure 2-1: Structure of a biological neuron

An important feature of biological neural networks is that they are fault tolerant, meaning that individual neurons can have failures, or even die, without a negative impact on the brain's performance.

An artificial neural network is made up of a number of basic processing units, called artificial neurons or simply nodes (the terms will be used interchangeably throughout the text). Each neuron is connected to other neurons by means of directed links with an associated weight representing the information being used to solve a problem. In its simplest form, an artificial neuron receives binary input signals, calculates a weighted sum of inputs and compares it to a threshold to determine the binary output.

The activity level of each neuron is a function of the inputs it receives, and its result is sent as a signal through connections to several other neurons. Each neuron can send only one signal at a time. The activity level of a neuron is adjusted by summing individual incoming signals, scaled down by the weight of the incoming connection which is defined by the activation function. This activation function is assumed to be nonlinear.

The three most often used forms of nonlinearities are hard limiting (either the step or the signum function), threshold and soft limiting (typically the sigmoidal function which scales the activity level, i.e. the output of the neuron, to a range between 0 and 1). Activation functions are discussed in more detail in Section 4.3.

Activation functions are specified by the researcher and vary from one neural network model to another. Generally the output of a neuron will be 0 until the activity level crosses some specified threshold value, at which stage it changes to 1.

The neuron's weights are adaptable. Positively weighted connections are known as excitatory while negatively weighted connections are said to be inhibitory. The adaptation of the weights is performed by a learning algorithm. This adaptation gives the system its capability to learn by an example and then generalize for new data. The concept of weight settings is analogous to the notion of memory in a conventional computer.

Figure 2-2 [Galkin, 2001] is a mathematical representation of an artificial neuron with I inputs X_i and corresponding weights W_i. T is the threshold level and O the neuron's output.

The nodes of an artificial neural network can be visualized as sets that are arranged in three (or more) layers: input nodes through which the network receives the values of the independent variables, a layer (or layers) of "hidden" nodes where the calculations are performed, and the output node(s) through which the network delivers its estimate of the values of the dependent variable(s). The outputs of one layer serve as the inputs to the next, as illustrated in Figure 2-3 which has one hidden layer.

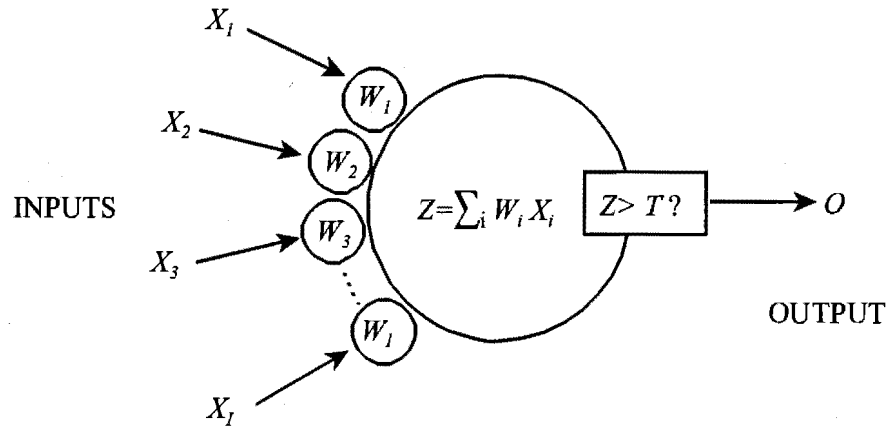

Figure 2-2: An artificial neuron

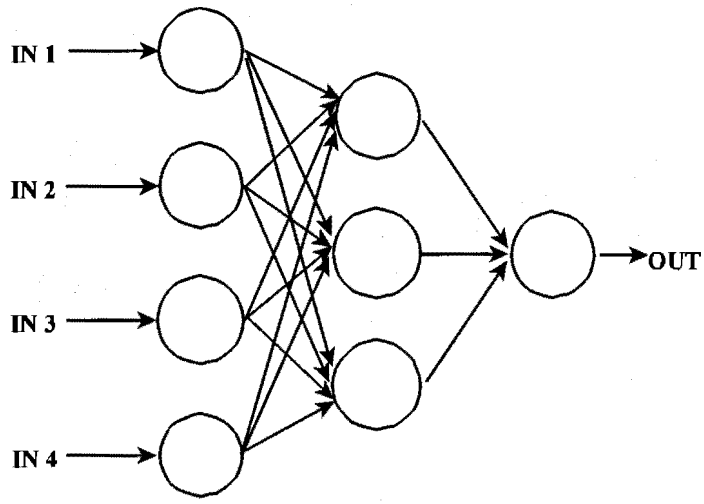

Figure 2-3: An artificial neural network

This configuration of an artificial neural network is called feedforward because, once the input layer neurons have values assigned to them, the network evolves layer by layer to determine its output. The dependence of the output values on the input values is complex as it includes all synaptic weights and thresholds. Usually this dependence does not have a meaningful analytic expression that can be mathematically interpreted. With artificial neural networks, this is invariably not necessary as the learning algorithms will, given the inputs, adjust the weights to produce the required output.

The key features of an artificial neural network, based on its comparison with the properties of a biological neuron, can be summarized as follows [Fausett, 1994]:

- The processing unit (node) receives several signals.
- Signals may be adapted by a weight (the value associated with a connection path between two processing units in an artificial neural network), similar to that of the chemical process at the receiving synaptic gap of a biological neuron.
- The processing unit sums the weighted inputs.
- The neuron transmits a single output after receiving sufficient input.
- The output may be transmitted to many other neurons.

More features of artificial neural networks that are suggested by biological neurons are:

- Information processing is local.
- Memory is distributed: "long-term" memory resides in the neuron's weights (synapses), while "short-term" memory corresponds to the signals sent by the neurons.

- A weight (synapse strength) may be modified by experience.
- Weights (neurotransmitters for synapses) may be excitatory or inhibitory.
- Artificial neural networks are fault tolerant in that they can be trained not to take small changes to the network into account, and can be re-trained in cases of massive damage, e.g. loss of data and/or some connections.

By the late 1980s, several organizations were already applying artificial neural networks to solve a variety of information-processing or pattern recognition problems in commerce and industry that have proved to be intractable or very difficult to solve with conventionally programmed digital computers.

Examples are cognitive tasks such as the visualization of speech phenomes (Teuvo Kohonen's "phonetic" typewriter [Kohonen, 1988]), learning to speak and understand a natural language (e.g. Kohonen's speech-recognition artificial neural network [Kohonen, 1990]) or guiding a mechanical hand to grasp objects of different shapes and consistencies, and optimization problems such as scheduling airline flights and allocation of seats between discount and standard fare classes to maximize airline's profit (e.g. the Airline Marketing Tactician developed specifically for this purpose, [Hecht-Nielsen, 1988]).

In many of the applications that are suitable to artificial neural network modelling it is virtually impossible to write down a series of logical or arithmetic steps even though it is possible to specify the tasks exactly and even develop numerous examples of the function being carried out.

One of the strong motives for the continued interest and enthusiasm in artificial neural networks is exactly this promise of solving a diversity of problems for which no algorithmic software exist, nor is likely to be developed for implementation on conventional computers. It is in cases like these that an experimental ad hoc approach based on heuristic methodology and quasi-

rigorous techniques of network learning to select the appropriate neural system for a specific application is necessary.

Scientists and technologists from a number of disciplines have been attracted to the field of artificial neural networks, for various reasons. For example computer scientists are interested in opportunities that are opened by the massively parallel computational networks in the areas of artificial intelligence, computational theory, modelling and simulation; mathematicians are fascinated by the potential of mathematical modelling applied to complex large systems phenomena; electrical and computer engineers are interested in building electronic integrated circuit-based intelligent machines and also looking at artificial neural systems as computing networks for signal processing; neuroscientists are interested in modelling biological neural networks, physicists envisage analogies between artificial neural network models and the nonlinear dynamic systems they study while psychologists view artificial neural networks as possible prototype structures of human-like information processing. The field is evidently strongly interdisciplinary.

2.4 Statistical modelling versus artificial neural networks

The fields of Statistics and artificial neural networks are in many respects closely related. Both disciplines developed methodologies with the aim to learn, or predict, from examples.

There are, however, still conflicting opinions among statisticians on the usefulness of artificial neural networks for statistical inference. Many are sceptical of the empirical approach of artificial neural network research where algorithms are developed for solving a particular application problem. This is in contrast with statistical research where implementation is often a secondary issue to

the theoretical assumptions underlying the analysis and is one of the main differences between Statistics and artificial neural network research methodology.

Another difference between the two is that Statistics has historically developed to deal with linear problems, while artificial neural networks are designed to specifically address nonlinearities where large volumes of data are available but little is known about the complicated relationship between the inputs and outputs. The application fields of the two methodologies are accordingly often different.

In 1993 the NATO Advanced Study Institute brought together for the first time more than 100 participants from both fields with a view to articulate differences and similarities in these fields, and to foster better cooperation amongst scientists and researchers. Some of the differences between the two approaches that have been noted by this group during the workshop are [Cherkassky & Wechsler, 1994]:

- Artificial neural networks mainly deal with very large data sets resulting in models that are of much higher complexity (in terms of the number of parameters) than statistical models which have usually been developed for smaller samples.

- In artificial neural networks the main objective is prediction or generalization. The complexity of the model often prohibits any interpretation or analytic expression. In Statistics the aim is usually interpretability of the model. This requires structured models such as linear regression and classification trees. Even in Statistics, however, structured models for high-dimensional problems may also be difficult to interpret due to the large model size.

- Artificial neural networks employ flow-through processing, an iterative process where only one sample is processed at a time, usually combined

with the slow computational algorithm of gradient descent (cf. Section 4.5). Most statistical methods use all the data to construct the model - a process known as batch-processing which is much faster.

- Artificial neural networks are readily understood and implemented by novice users as they are computationally simple, albeit slow and at the expense of multiple presentations of the data. There is also the perception that they require little human expertise. Statistics, on the other hand, use complex methods often with underlying assumptions about the data that are perceived as difficult to understand and to use, therefore statistical methods tend to be less popular with non-statisticians.

- Artificial neural network methods seem to be more robust than statistical methods. Reasonable results are produced by artificial neural networks even with suboptimal choices of parameters such as network size, learning rate and initial weights. The quality of solutions produced by artificial neural networks can however not be guaranteed as in Statistics where confidence intervals for estimates are routinely provided.

It is clear that no single method predominates for all possible data sets. Both artificial neural networks and statistical methods perform asymptotically satisfactorily as both produce reasonable estimates for large samples. For ill-posed problems with sparse data sets, however, asymptotic performance is irrelevant and the best method is the one that conforms to the properties of the data at hand.

Participants at the workshop mentioned above concluded that the real research problem is not to determine a single "best" method, but to characterize the class of functions or mappings, in conjunction with assumptions about properties of the data such as noise and smoothness, for which a given method perform best. This view is still held amongst scientists and researchers today.

Chapter 3

Historical development of artificial neural networks

By 1930 it was widely accepted that the mind resides in the brain, which is packed with neurons. The general nature of synapses and the threshold response of neurons were also known by then. Scientists began their search for an explanation of brain functions such as memory, perception and reasoning in terms of brain mechanisms and structures such as neurons, synapses and thresholds. This has led directly to the development of artificial neural systems.

In this chapter the evolution of artificial neural networks since the early 1940s is traced, starting with the work of Warren McCullogh and Walter Pitts [McCulloch & Pitts, 1943].

Artificial neural networks initially evoked considerable interest amongst scientists, but enthusiasm waned when some fundamental limitations of these systems were evinced by Marvin Minsky and Seymour Papert in 1969 [Minsky & Papert, 1988 c.1969].

In 1986 the discovery of successful extensions of artificial neural network knowledge by James McClelland, David Rummelhart and the PDP (Parallel

Distributed Processing) Group brought about another revitalization of this field [McClelland & Rumelhart, 1986], to the extent that the study and application of artificial neural networks now encompass many disciplines such as engineering, physics, mathematics, neuroscience, medicine, psychology and finance.

Only a synoptic overview of the historical development of artificial neural networks is provided here. It is neither possible nor the intention to afford an exhaustive bibliography in this thesis. Rather, the most important contributions over the past decades were singled out for their impact on the evolution of the field of artificial neural networks.

3.1 Warren McCullogh and Walter Pitts

Warren McCullogh and Walter Pitts explored the computational capabilities of network models with a very simple design during the middle decades of the previous century. Their pioneering publication, "A Logical Calculus of the Ideas Immanent in Nervous Activity" in which they outlined the first formal model of an elementary computing neuron in 1943, is generally regarded as the genesis of the development of artificial neural network systems [McCulloch & Pitts, 1943]. In this paper McCullogh and Pitts presented the first sophisticated discussion of "neuro-logical networks" and stated the doctrine and many of the fundamental theorems of their axiomatic representation of neural elements explicitly. The paper caused considerable excitement amongst scientists and spurred a flurry of interest in artificial neural network systems.

The McCullogh-Pitts abstract model of a neuron is characterized by a finite number I of inputs X_i, multiplicative weights W_i which are either excitatory (i.e. the weight $Wi = +1$) or inhibitory (i.e. the weight $W_i = -1$), a threshold

level T and an output O. The inputs and outputs assume the binary values 0 or 1. The threshold can be any positive integer. Mathematically the output of a McCullogh-Pitts neuron can be expressed in terms of its inputs by

$$O = f(\sum_{i=1}^{I} W_i X_i - T) \tag{3.1}$$

where $f(p) = 0$ if $p < 0$, and $f(p) = 1$ if $p \geq 0$, i.e. the neuron "fires" when the total excitation it receives reaches or exceeds the specified threshold value. The threshold term T can be eliminated from this equation by simply adding an extra input connection from a node with its value fixed at 1 and weight the negative of the threshold value T. This has the advantage that the threshold value T can be adjusted along with the other weights. An external input can be supplied to the network in a similar way by adding an extra term to the sum of some of the inputs.

The physical assumptions of the network, as stated in the original paper and reproduced in "Embodiments of Mind" are ([McCulloch, 1965], p.22)

1. The activity of the neuron is an "all-or-none" process.
2. A certain fixed number of synapses must be excited within the period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron.
3. The only significant delay within the nervous system is synaptic delay.
4. The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time.
5. The structure of the net does not change with time.

Even though this model is very simplistic, and memory is essentially ruled out under these conditions, it has substantial computing power.

It was believed at that time that it is always possible to construct a McCullogh-Pitts network which will be capable of representing whatever input-output configurations might be realized by a system with an arbitrary memory mechanism, provided that activity is allowed to persist in the network. In actual fact, it can perform the basic logic operations NOT, OR and AND - as well as any combinational function using either the NOT and OR, or the NOT and AND - provided that the weights and thresholds are appropriately selected.

At this time, the idea of constructing devices out of simple logical elements with neuron-like properties was by no means novel. Turing, for example, published a paper on this topic in 1936 [Turing, 1936–37] where he described an abstract representation of a computing device.

A Turing machine consists of a read/write head that scans a two-dimensional tape divided into squares, each of which is inscribed with a 0 or a 1. It moves and writes using a table of instructions (one for each state and binary input) known as the functional states of the machine. A Turing machine is therefore more like a computer program (software) than a computer (hardware) and was an attempt by Alan Turing to provide a mathematically precise definition of an algorithm.

The development of stored-program digital computers by Von Neumann and others [Burks & Von Neumann, 1947] lent further impetus to research in this field in the 1940s. After the publication of Rashevsky's book "Mathematical Biophysics" in 1938 [Rashevsky, 1938] a group of mathematical biophysicists at the University of Chicago got together. The aim was specifically to investigate "nerve nets" consisting of formalized neurons and connections that might be able to perform psychological functions. Amongst those who made innovative contributions to this field were Pitts, McCullogh, Landahl and Householder ([Pitts, 1942], [Pitts, 1943], [Landahl & Pitts, 1943], [Householder & Landahl, 1945]).

It has been pointed out that McCullogh and Pitts's major contribution to the field of artificial neural networks was not so much the schematic construction of neural circuits specified by their conditions of firing, as these have previously been used diagrammatically to illustrate simple reflex arcs. It was instead their broader vision of conceptualizing the brain as a "computing machine", as put so succinctly by Seymour Papert in his introduction to the text of *Embodiments of Mind*: "The step that needed boldness of conception and mathematical acumen was the realization that one could formalize the relations between neurons well enough to allow general statements about the global behaviour of arbitrarily large and only partly specified nets to be deduced from assumptions about the form and connectivity of their components." ([McCulloch, 1965], p.xvii). McCullogh and Pitts were the first researchers to provide a set of mathematical instruments that was powerful enough to describe neurophysiological hypotheses about brain mechanisms.

The McCullogh-Pitts paper laid the groundwork for the possibility of formulating more precise and particular hypotheses as explored by themselves and many others in the following years and decades ([Pitts & McCulloch, 1947], [Hebb, 1949], [Rosenblatt, 1958], [Widrow & Hoff, 1960], [Landahl, 1961], [McCulloch, 1962], [Rosenblatt, 1962], [Widrow, 1962], [Cover, 1965], [Minsky & Papert, 1988 c.1969], [Hopfield, 1982], [Hopfield, 1984], [McClelland & Rumelhart, 1986]). In their essay "How We know Universals", for example, McCullogh and Pitts described network architectures which were in principal capable of recognizing spatial patterns in a manner invariant under groups of geometric transformations [Pitts & McCulloch, 1947]. (The architecture of a network refers to the arrangement of the neurons in layers and the corresponding pattern of connections. Network architectures are discussed in more detail in Chapter 4, Section 4.2.)

The popular intellectual movement called cybernetics, which attempted to combine concepts from biology, neurology, psychology, engineering and mathematics, emerged from these ideas. The field attracted a lot of interest from those researchers, especially psychologists and computer scientists, who strived to gain a deeper understanding of the brain in a manner that would also be mathematically simple enough to allow theoretical analysis.

Initially, research focused largely on localization and specific artificial neural network configurations, called architectural schemes or the network's topology, that could perform specific functions were developed. A well-known and often quoted example in the literature is the work of Lettvin et al. on the physiology of vision [Lettvin & Pitts, 1959]. The goal, however, soon changed to building machines that could learn ([Minsky, 1954], [Nillson, 1990]). As the simple concept of reinforcement learning was already well-known in behaviouristic psychology, most of the early experiments used a reinforcement-based network learning system. A reinforcement-based network learning system must be capable to generate a sufficient variety of actions from which to choose, as well as some criterion of relative success.

3.2 Donald Hebb

The psychologist Donald Hebb designed the first learning law for artificial neural networks. In "The Organization of Behaviour", published in 1949 [Hebb, 1949], he proposed a learning scheme for updating neurons' connections that had a considerable impact on future developments in the field. His was the first attempt to base a large-scale theory of psychology on suppositions about artificial neural networks.

Based on the biological discovery that a synapse's resistance to an incoming signal is changed metabolically during a "learning" process, Hebb showed that

networks might learn by storing information in connections by constructing so-called cell-assemblies: subfamilies of neurons which are frequently activated together become linked into a functional organization and thus learn to support each other's activities. This is referred to as the Hebb learning rule and forms the basis for the concept of associative memory. The Hebb learning rule is described in more detail in Section 4.4.1.1.

3.3 Marvin Minsky

The first reinforcement-based neurocomputers which adapted connections automatically were built and tested by Marvin Minsky during the 1950s. His original machine, built in 1951, consisted of electronic units interconnected by a network of links. These links had adjustable probabilities of receiving activation signals and then transmitting them to other units. Learning was by means of a reinforcement process in which each positive or negative judgement about the machine's behaviour was translated into a small change in the probabilities associated with the corresponding connections. In his Princeton PhD dissertation in Mathematics he postulated many new theories and theorems about learning in artificial neural networks, secondary reinforcement, circulating dynamic storage and synaptic modifications [Minsky, 1954].

It should be borne in mind that modern computers as we know them today did not exist in those days. The concept of programming had barely appeared at that time. It was an era in which Thomas Watson in 1943, then chairman of IBM, could blithely state "I think there is a world market for maybe five computers". However, with the advance of modern computers it became much more viable to experiment with different learning schemes, as well as to do research based on learning. One example is Arthur Samuel's research on programming computers to learn to play checkers using a success-based reward

system. He was able to use a variety of error-correcting vector addition procedures that was later developed to utilize more complex interactions between the partial predicates [Samuel, 1959], [Samuel, 1967].

3.4 Frank Rosenblatt

Frank Rosenblatt formally introduced the neuron-like element called a perceptron towards the end of this decade. In his 1958 paper [Rosenblatt, 1958] and subsequent book [Rosenblatt, 1962] he criticized the lack of randomness and the inflexibility of existing artificial neural network models compared to biological neural networks. His research investigated a simple brain model emulating the physical structures and neurodynamic principles which underpin intelligence.

Various different types of brain models had so far been proffered by scientists ranging from philosophers, psychologists, biologists and mathematicians to electrical engineers ([Hebb, 1949], [Minsky, 1954], [Von Neumann, 1958]). Rosenblatt unique contribution was that he proposed a theory of statistical separability based on probability theory, rather than symbolic logic, to develop a class of network models known as perceptrons and formulated his Perceptron Convergence theorem ([Rosenblatt, 1962], pp.109-116).

Earlier applications of probability theory to brain models include the paper by Landahl, McCullogh and Pitts ([Landahl & Pitts, 1943]). Their approach differs from Rosenblatt's in that the impulses are assumed to be propagated with known frequencies but with uncertainties in their timing, while the topology of the network is still assumed to be a strictly deterministic organization which is fully known. They formulated a theorem to obtain the expected frequencies with which different cells will respond. Other work which attempted to develop statistically organized networks which are characterized

by probability distributions for thresholds, synaptic types and origins of connections include that of Shimbel and Rapoport [Shimbel & Rapoport, 1948]. Uttley also described a reward-modified machine, based on statistical models and using conditional probabilities, which was a predecessor of the perceptron [Uttley, 1956], [Uttley, 1959a], [Uttley, 1959b].

Although the term perceptron was originally intended as a generic name for a variety of theoretical neural net models, it was widely used to describe a trainable machine (hardware) capable of learning to classify certain patterns by modifying connections to the threshold elements.

In its simplest configuration the perceptron is formed as three layers of signal generating units, or neurons: the sensory inputs which are connected to an "association" layer on a partial and random basis, which in turn is randomly connected to a response layer of neurons which produce the outputs of the network; these response neurons inhibit each other and those association neurons from which they do not receive input.

The logical properties of a perceptron are defined by the connections among the neurons (the topological organization), a set of signal processing functions, i.e. the rules dictating the generation and transmission of signals, and a set of memory functions, the rules regulating modification of the network properties as a consequence of activity. By including a mechanism for change in the neural net, Rosenblatt aimed to achieve a model for memory and learning.

The perceptron learning rule's iterative weight adjustment makes it more powerful than the Hebb rule. The rule is formulated more formally in mathematical terms in Section 4.4.2.1.

Rosenblatt demonstrated that the perceptron can generalize, i.e. when presented with similar inputs - even novel ones - it will give the same response, as well as learn. The system is sufficient for pattern recognition, associative learning, as well as such cognitive sets that are necessary for selective recall.

However, as soon as the response calls for the recognition of a relationship between stimuli, the problem becomes too difficult for the perceptron. Rosenblatt acknowledged that statistical separability alone does not provide a sufficient basis for higher order abstraction and that a more advanced system seems to be needed ([Rosenblatt, 1958], p.405).

The perceptron used different mathematical learning systems that could be roughly divided into two categories [Smith, 1993]. In the first learning category, the association neurons that continue to be active while the response is given gain in strength, with the result that the response neurons become increasingly sensitive to the input patterns that they initially responded to. This corresponds to the so-called self-organization or competitive learning models currently used. The other learning category was known as forced learning as information from outside the perceptron activates the appropriate response neuron when it is presented with a specific input pattern. The response neuron thus becomes more sensitive to the input pattern. This process is simply reinforcement learning as mentioned earlier.

Rosenblatt's idea fired the imagination of scientists and engineers alike and laid the groundwork for the basic machine learning algorithms still used today. At this stage the field became widely known as Connectionism because the weighted connections between neurons essentially contain the information in the system and therefore determine the behaviour of these networks.

3.5 Bernard Widrow and Marcian Hoff

During the early 1960s another powerful learning rule, called the Widrow-Hoff learning rule, was developed by Bernard Widrow and Marcian Hoff ([Widrow & Hoff, 1960], [Widrow, 1962]). (The Widrow-Hoff learning rule is also referred to as the least mean squares (LMS) rule in technical literature.)

The ADALINE (ADAptive LInear NEuron), which is explained in more detail in Section 4.4.2.2, is not a network but a single neuron that produces an output based on a pattern of inputs, like the perceptron. The rule is closely related to the perceptron learning rule. While the perceptron rule adjusts the connection weights to a unit whenever the response of the unit is incorrect, the ADALINE's learning method incorporates supervised learning (cf. Section 4.4.2) where the network is given feedback indicating not only whether the output is incorrect, but also what the output should have been. The Widrow-Hoff rule adjusts the weights to reduce the difference between the network's output and the target output. It achieves this by descending the gradient in the error surface, i.e. minimizing the summed squared error during training.

Even though the difference between the two rules (perceptron and Widrow-Hoff) are small, the Widrow-Hoff rule leads to an improved ability of the network to generalize, i.e. to respond to input that is similar, but not identical, to that on which it was trained. The Widrow-Hoff learning rule for a single-layer artificial neural network is a precursor of the backpropagation rule that is used for many multilayer artificial neural networks (cf. Section 4.5).

Early applications of ADALINE and its extension to MADALINE (for MAny ADALINES) include pattern classification, weather forecasting and adaptive controls. Since the ADALINE is a linear neuron, it's applications are limited to learning linearly separable classes.

In spite of the enthusiasm and successes of this period, more complex computational problems could not be solved by these early machine learning theorems and artificial neural network research entered a stagnation phase. The relatively modest computational resources available to researchers also contributed to the slowdown into artificial neural network research. Nils Nilsson's monograph on learning machines provide a clear summary of many of the developments and obstacles of that time [Nillson, 1990].

3.6 Marvin Minsky and Seymour Papert

The final publication of this era was the book "Perceptrons" by Marvin Minsky and Seymour Papert in 1969 [Minsky & Papert, 1988 c.1969]. In this text Minsky and Papert evaluated the perceptron as the simplest learning machine, i.e. as a class of computations (parallel-machine architectures) that make decisions by weighing evidence. Up to this stage many experiments with perceptrons have taken place, but nobody has been able to satisfactorily explain why perceptrons were able to learn to recognize certain kinds of patterns but not others. Minsky and Papert revealed some fundamental limitations of loop-free connectionist learning machines and proved that one-layer perceptrons were incapable of learning to distinguish classes of patterns that were not linearly separable, using the well-known logical EXCLUSIVE-OR (XOR) function to illustrate the weakness of the perceptron.

The XOR function is a logic function that is not symmetric in its treatment of the two input values. The response is "true" if one of the input values is "true" and the other input value is "false"; otherwise the response is "false". Using a bipolar representation of the logical input and response values, the four input target pairs are:

$$\begin{array}{lcl} (x_1, x_2) & \rightarrow & y \\ (+1, +1) & \rightarrow & -1 \\ (+1, -1) & \rightarrow & +1 \\ (-1, +1) & \rightarrow & +1 \\ (-1, -1) & \rightarrow & -1 \end{array}$$

Inspection of the graphical display in Figure 3-1 shows that the XOR function is not linearly separable.

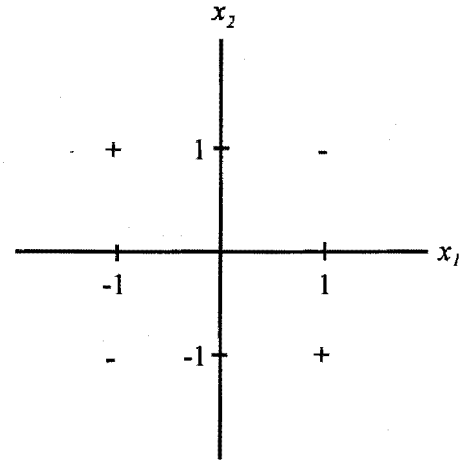

Figure 3-1: Response for the logic XOR function

Interest in the field of artificial neural networks waned considerably when the limited potential of layered learning networks was thus exposed. Even though Minsky and Papert's text has the reputation of dampening the research interest in artificial neural networks at that time, the authors argue (in their Epilogue to the 1988 reprint, [Minsky & Papert, 1988 c.1969]) that their intent was not so much to publicize the limitations of perceptrons as to critically evaluate their abilities and foster an understanding of the general principles governing their capabilities. The goal of the book was, according to their Introduction ([Minsky & Papert, 1988 c.1969], p.1), to offer general insights into the related fields of parallel computing, pattern recognition, knowledge representation and learning. The text was aimed at specialists in these fields, but they also strived to reach an audience who was interested in the general theory of computation in order to stimulate interest in the theory of genetic programming and the capabilities of parallel-network learning machines.

3.7 During the 1970s and early 1980s

Dubbed the "Quiet Years" by Laurene Fausett, further pioneering work in the field of artificial neural networks was accomplished by only a handful of researchers during the next ten to fifteen years [Fausett, 1994].

3.7.1 Japanese scientists

Of the important contributors mentioned by various authors ([Zurada, 1992], [Smith, 1993], [Fausett, 1994], [Haykin, 1999]) were the Japanese scientists, Sun-Ichi Amari who studied the mathematical theory of artificial neural networks and learning in networks of threshold elements ([Amari, 1972], [Amari, 1977]) and the development of the class of artificial neural network architectures specializing in character recognition known as neocognitrons by Kunihiko Fukushima [Fukushima & Miyaka, 1980].

3.7.2 Kohonen

Another important activity was the demonstration of self-organizing maps using competitive learning, motivated by topologically ordered maps in the brain ([Von der Malsburg, 1973], [Willshaw & Von der Malsburg, 1976]). Teuvo Kohonen of Finland worked out a theory of associative memory where pairs of patterns are stored so that presentation of one of the patterns in a pair directly evokes the associated pattern without any serial search [Kohonen, 1977]. An associative memory is therefore content-addressable. Kohonen also developed a model of self-organizing topographical maps [Kohonen, 1984].

3.7.3 Anderson; Grossberg and Carter

James Anderson's "Brain-State-in-a-Box" associative artificial neural networks [Anderson *et al.*, 1977] should also be mentioned as well as the extensive work

done by the American Stephen Grossberg and his colleagues who studied in detail experimental evidence of cognitive tasks and used computer-based simulations to characterize the behaviour of models they developed. Together with Gail Carpenter he developed a special class of self-organizing artificial neural networks based on the theory of adaptive resonance (ART) [Grossberg, 1977].

3.7.4 Simulated annealing and the Boltzmann machine

Simulated annealing is a procedure for solving combinatorial optimization problems by reducing the likelihood of an artificial neural network to become trapped in a local minimum which is not a global minimum, described in 1983 by Kirkpatrick et al. [Kirkpatrick & Vecchi, 1983]. The Boltzmann machine, a class of artificial neural networks used for solving constrained optimization problems, was developed in 1985 by Ackley, Hinton and Sejnowski as a stochastic learning algorithm, using the concept of simulated annealing [Ackley *et al.*, 1985].

3.8 John Hopfield

The resurgence of interest in artificial neural network models started when the physicist John Hopfield introduced a recurrent artificial neural network architecture for associative memories in the early 1980s ([Hopfield, 1982], [Hopfield, 1984]). He also introduced the concept of an energy function of the weights and activations of the artificial neural network. This function always decreases or remains unchanged with each iteration of the evolving artificial neural network, thus providing an elegant solution to the tough problem of artificial neural network convergence to a stable state.

3.9 James McClelland and David Rummelhart

The discovery of successful extensions of artificial neural network knowledge, however, had to wait until 1986 when another revitalization of this field came about after the publication in 1986 of two volumes on parallel distributed processing, written by James McClelland, David Rummelhart and fourteen collaborators known as the PDP (Parallel Distributed Processing) Group. The announcement of the discovery of a method that enables an artificial neural network to learn to discriminate between classes of patterns that are not linearly separable heralded an "connectionist revolution" as it revived enthusiasm in the field of artificial neural networks and almost immediately spurred an intense growth of activity in artificial neural network research.

The authors introduced a new learning rule and concepts for more complex multilayer systems which exploit the previously underestimated computing power of layered networks. The "Generalized Delta Rule" "provides a direct generalization of the perceptron and ADALINE learning procedures which can be applied to arbitrary networks with multiple layers and feedback among layers"([McClelland & Rumelhart, 1986], vol.1, p.113), as opposed to Rosenblatt's perceptron which operates only on the connections between one layer of input units and a single output unit. Also known as backpropagation, this recursive error propagation algorithm trains an artificial neural network to approximate virtually any function, including arbitrarily complex nonlinear functions, thereby removing one of the most essential network training barriers that grounded the mainstream efforts of the mid-1960s. Backpropagation, which is employed by most popular artificial neural networks today, is essentially a nonparametric statistical modelling technique in which the shape of the relationship between inputs and outputs is determined by the data. This algorithm is be explained in more detail in Section 4.5.1 on artificial neural network models.

It is now widely acknowledged ([Zurada, 1992], [Smith, 1993], [Fausett, 1994], [Anderson, 1995], [Bishop, 1995], [Haykin, 1999]) that the mathematical framework for this new training algorithm for layered networks was already formulated by Paul Werbos in his Harvard Ph.D. thesis in 1974 [Werbos, 1974], although it went largely unnoticed at that time.

3.10 Current state of affairs

After the publication of McClelland and Rummelhart many new artificial neural networks research programs were initiated. One of the more sensational projects was the development of a backpropagation artificial neural network model called NETtalk, developed by Charles Rosenberg and Terry Sejnowski in 1986 [Smith, 1993]. The NETtalk artificial neural network translates written English into a phonetic representation that can be used to produce machine speech, i.e. it trains the network to read out aloud - an impressive demonstration of the capabilities of artificial neural networks.

The Boltzmann machine, developed in 1985 by Ackley, Hinton and Sejnowski, laid the groundwork for the development of sigmoid belief networks by Neal ([Ackley *et al.*, 1985], [Neal, 1992]). These networks accomplished a significant improvement in the learning performance of artificial neural networks [Haykin, 1999]. The use of mean-field theory by Saul et al. further improved the learning performance of sigmoid belief networks [Saul & Jordan, 1996].

Radial basis function (RBF) artificial neural networks, an alternative design for layered feedforward networks, was described in 1988 by Broomhead and Lowe [Broomhead & Lowe, 1988]. The RBF is designed to detect local regions (clusters) in the input feature space. It operates by measuring the distance between the input vector and the centre of each of its basis functions.

Vladimir Vapnik and his coworkers invented a computationally powerful class of supervised learning networks, called support vector machines, for solving pattern recognition, regression and density estimation problems in the early 1990s ([Cortes & Vapnik, 1995], [Vapnik, 1995]). This method is based on results in the theory of learning with finite sample sizes. The Vapnik-Chervonenkis (VC) dimension, which provides a measure for the capacity of an artificial neural network to learn from a set of examples, is a key feature of support vector machines.

The intensity of research in the neurocomputing discipline can be measured by the quickly growing number of conferences and journals devoted to the field. The first International Conference on Neural Networks was held in 1987 under the auspices of the IEEE. At that conference the International Neural Network Society was organized. The International Joint Conference on Neural Networks is now held every year, bringing together researchers from all over the world and representing various disciplines. A host of other workshops and conferences are held annually or biannually, as evidenced by simply typing the phrase "neural network" into any internet search engine.

Many edited volumes that include collections of papers and numerous books have appeared and many applications that could be solved by artificial neural networks had expanded from small test-size examples to large practical tasks, e.g. [Cottrell, 1990], [Hecht-Nielsen, 1990], [White, 1992], [Zurada, 1992], [Smith, 1993], [Cherkassky & Wechsler, 1994], [Fausett, 1994], [Bishop, 1995], [Hewitson & Crane, 1994], [Anderson, 1995], [Cohen, 1995], [Hassoun, 1995], [Martin, 1995], [Vapnik, 1995], [Venkatasubramanian & Rengaswamy, 1995], [Mira & Sandoval, 1995], [Smolensky & Rumelhart, 1996], [Mitchell, 1997], [Anthony & Bartlett, 1999], [Fine, 1999], [Kay & Titterington, 1999] and [Haykin, 1999].

Artificial neural networks have been shown to be highly effective and robust in many practical problems, encompassing many disciplines such as engineering, physics, mathematics, neuroscience, medicine, psychology and finance. They have proved to be especially effective in pattern recognition and signal processing.

Some of the interesting examples of pattern recognition are the automatic recognition of handwritten characters such as zipcodes [LeCun *et al.*, 1989]; learning to recognize spoken words [Lang & Hinton, 1990] and learning to recognize faces [Cottrell, 1990]. One of the first commercial applications of signal processing was to suppress noise on a telephone line [Fausett, 1994]. Fausett cites a number of other interesting artificial neural network applications, e.g. J. A. Anderson's "Instant Physician" which will provide a medical diagnosis and treatment when presented with a set of symptoms and a mortgage assessment network that was commercially developed by E. Collins, S. Ghosh and C. L. Scofield. Another medical application is EEG spike detection using an artificial neural network classifier to aid neurologists in the detection of abnormal brain waves which indicate an imminent epileptic seizure, developed by R. C. Eberhart and R. W. Dobbins [Eberhart & Dobbins, 1990]. As a financial application, A. F. Refenes et al. developed a neural model to assess stock performance [Refenes & Francis, 1994].

The quest for a general solution to all learning problems continues, but at least a better understanding of which types of learning processes are likely to work on which classes of problems has evolved over the years of research into this field.

Chapter 4

Artificial neural networks

Various different types of artificial neural networks, each with its own characteristics, have evolved that are suitable to different applications. Careful consideration should therefore be given to the selection of the neural processing tools, the model structure and the initial conditions when selecting an artificial neural network model to fit to your data. A network is characterized by the following features:

- its pattern of connections between the neurons (architecture);
- its training or learning algorithm, i.e. its method of determining the weights between the connections;
- its activation function which determines its output.

The objective of an artificial neural network model is to locate the global solution in a complex problem domain where a number of suboptimal solutions may exist. Each processing element (neuron) performs a simple task. The power of artificial neural networks come from the collective behaviour of the neurons in a network as it is the connections between these neurons that give an artificial neural network the ability to learn patterns and interrelationships

in the data. The weights between connections contain the knowledge of the artificial neural network and training is the process of adjusting these weights. This is a highly complex parallel process whose features cannot be reduced to phenomena taking place within individual neurons.

Artificial neural networks learn the desired input-to-output mapping by minimizing the error between the predicted and actual outputs. The model that produces the least possible error when data are passed through it, is considered the best. An error domain graph, plotting the weights of an artificial neural network against the error produced, is useful to describe the error. An evolving artificial neural network will eventually reach a state where all neurons continue working but no further changes in their state occur. A network may have more than one stable state which is determined by the choice of synaptic weights and thresholds for the neurons. The aim is to configure a network that does not stop training at one of the suboptimal solutions or a local minimum.

The first step involved in the generation of a neural model is the collection, auditing and preprocessing of the data. This aspect is discussed in more detail in Section 4.1.

Following the data preparation stage is the generation of the neural model. The stages involved in the generation of a neural model include designing the model, optimizing the model, training the model - including the validation of the model - and finally testing the model before implementation of the final model.

Network topologies

Network topologies can vary from single model designs to complex designs that partition the problem into subgroups.

Partitioned topologies is one of the options available when, after developing a single artificial neural network system and experimenting with the parameters, it becomes apparent that the system is not performing satisfactorily. It entails partitioning the input data into subsets, each of which is then processed by a single neural model, resulting in a network ensemble. This technique can yield better results than those obtained from single neural models if executed carefully.

The main difficulty faced when developing a partitioned topology is the selection of input fields for the subsets: if variables that should be assigned to the same subset is split between two or more groups the performance of the neural system will be degraded. Simple correlation coefficients or simple scatterplots for each input variable against all the other variables can be generated to aid detection of correlated variables (thus belonging to the same subset), but problems arise in practice when there are a large number of input fields. For example, 50 input fields translate to $\binom{50}{2} = 1225$ scatterplots that have to be generated and inspected. This problem can be partially overcome by applying domain knowledge. Applying more sophisticated statistical techniques, such as principal components analysis or factor analysis, to the data may also be useful to identify meaningful subsets of variables.

The focus here will be on single model designs as the extension to neural network ensembles follows logically.

The options available when developing a single model network are the type of model, the internal configuration, i.e. the architecture, and the initial conditions. The type of model is to a large extent determined by the research problem. Different artificial neural network models are presented in Sections 4.4 and 4.5. The architecture of the network, discussed in Section 4.2, refers to its associated artificial neuron connection set while the initial conditions are invariably determined by the researcher in a heuristic manner.

Training

During the training phase the known input-target pattern pairs, also called examples, are presented to the network and its weights are adjusted to produce the required outputs. One full pass through the training set is termed an epoch.

Typically, training continues until a preset condition is met. This may be the minimization of a predefined error function such as the residual error between the actual and the target outputs (achieved by minimizing the total sum of the squared differences between the target and the computed output nodes over all training patterns) or until a set number of epochs has been reached.

A host of different training or learning algorithms have been developed for different classes of problems. The most commonly used algorithms are discussed in more detail in Section 4.4.

Training an artificial neural network is, in most cases, an exercise in numerical optimization of a usually nonlinear function. The basic problem of optimization is to arrive at the best decision for a given set of circumstances, usually by minimizing certain cost functions defined by the user.

A huge body of literature exists on the subject in fields such as numerical analysis, operations research and statistical computing, e.g. [Bertsekas, 1995a], [Bertsekas, 1995b] and [Gill *et al.*, 1981]. No single best method for nonlinear optimization exists. For functions with continuous second derivatives, which would include feedforward nets (cf. Section 4.2.2.3.1) with the most popular differentiable activation functions (cf. Section 4.3) and error functions, it has been found that the various conjugate-gradient algorithms, including backpropagation (cf. Section 4.5) which is the most commonly used learning algorithm for feedforward artificial neural networks, are efficient in locating local optima. For global optimization, a number of approaches can be followed. Often simply running any of the local optimization methods from numerous

random starting points yield satisfactory results. Another alternative is to increase the number of hidden nodes in a feedforward multilayer artificial neural network (cf. Section 4.5). Caution should be exercised though, as too many hidden nodes for the problem may result in the system becoming too specific or overtrained (cf. Section 4.1.1.1). More complicated methods specifically designed for global optimization include simulated annealing, mean field tunneling ([Anderson, 1995], [Haykin, 1999]), genetic algorithms, particle swarm optimization and LeapFrog (e.g. [Ismail & Engelbrecht, 2000], [Snyman, 1983], [Van den Bergh & Engelbrecht, 2000]).

Testing and implementation

Testing the model includes a recognition phase when the weights are fixed, patterns are again presented to the network and it recalls the outputs. In the recall mode, i.e. the proper processing phase of an artificial neural network, the information is retrieved from the data.

The basic forms of neural information processing can be summarized as autoassociation, heteroassociation and classification [Zurada, 1992]. In the process of autoassociation, the artificial neural network is presented with a pattern similar (but possibly degraded) to a member of a set of stored patterns that the network has been trained on with the aim to associate the input with the closest stored pattern. When associations between pairs of patterns are stored in a net for recall, the association process is known as heteroassociation. Classification takes place when the set of input patterns are divided into a number of classes or categories. The network classifier is supposed to recall the information regarding class membership of the input pattern when presented with an input pattern.

The ability of an artificial neural network to correctly classify input data patterns that it has not been trained with (i.e. respond to patterns which are

similar, but not identical to training data) is termed generalization. Invariably the ultimate goal of the researcher is to be able to present novel patterns to the network for prediction or classification, in which case good generalization is important.

Various techniques have been designed to ensure that the neural system can be relied on to process real data. If the results indicate that the system performance is unacceptable, the developer must decide on one of the various options available: change the artificial neural network topology, use a different neural model, return to the data preprocessing stage and re-evaluate techniques used to manipulate the data, or even collect additional data fields that may provide more information on the particular problem.

It must be borne in mind that enough predictive information is needed to yield acceptable results. If not, it may be possible to partition the problem into smaller subproblems to try and achieve acceptable results.

4.1 Data

Data form the most important part in the development of an artificial neural network, therefore a thorough understanding of the data is essential. The collection, auditing and preprocessing of the data is a vital aspect in the development of any neural system and is analogous to that of any statistical project. An adequate volume of relevant, high quality data containing information about the behaviour that one is trying to model is essential during the artificial neural network's training phase. As in Statistics, it is important that the sample is representative of the problem in order for the network to be reliable. The old adage of "Garbage in, garbage out" is particularly appropriate for artificial neural networks!

The sampling units or observations are termed examples, patterns or samples in artificial neural network terminology. What is commonly known in Statistics as independent variables, explanatory variables or predictors in a regression-type setup, is called input variables or features in artificial neural network jargon, while the dependent variable is known as the target variable (as opposed to the output variable which is computed by the artificial neural network model, analogous to the predicted values in statistical modelling).

4.1.1 Data collection and auditing

Not only should records in the database accurately represent the information that is needed to build the artificial neural network, there should also be a sufficient amount of information available for the network to learn the input to output mapping in order to train and test an artificial neural network.

The amount of data required for an artificial neural network is one of the most important issues to be addressed during the data collection phase, and is related both to the topology of the neural model and the complexity of the problem to be modelled. The VC-dimension, one of the most important results in statistical learning theory as specified by Vladimir Vapnik and A. J. Chervonenkis, helps to quantify the difficulty when learning from examples. It relates training set size, architecture and generalization performance ([Cherkassky & Wechsler, 1994], [Vapnik, 1995], [Cherkassky, 1996], [Vidyasagar, 1997]). Larger training samples will allow the network to continue training longer or the use of more parameters, i.e. hidden nodes, in the model (cf. Section 4.2.2). This may improve the accuracy with which the artificial neural network can model complex functions [Smith, 1993]. It should be noted, though, that a critical training set size exists which, if exceeded, may have a negative effect on neural network performance. Research conducted in this area include active learning, an approach where the sample

size is selected during training, e.g. [Engelbrecht, 1999]. In the SPSS Neural Connection package it is proposed that the number of training data records (cf. Section 4.1.4) should be at least ten times the total number of free parameters or weights within the neural model [SPSS, 1995]. Galkin advocates at least 30 times as many training cases as weights in the network to avoid undertraining [Galkin, 2001]. These heuristics are guidelines only and should not be treated as rules.

4.1.1.1 Underfitting and overtraining of an artificial neural network

One of the critical issues in developing an artificial neural network is its ability to generalize, i.e. the network's ability to classify novel patterns that are not in the training set. This is the artificial neural network analogy to statistical inference.

Artificial neural networks, like other flexible nonlinear estimation methods such as kernel regression and smoothing splines, can suffer from either underfitting or overfitting. A network model that is not sufficiently complex can fail to detect fully the signal in a complicated data set, leading to underfitting. On the other hand, a network that is too complex may fit the noise, not just the signal, leading to overfitting. Overfitting, or overtraining, occurs either when too few records are used for training, or when the network is left to train for too long, resulting in the network learning the training examples with zero error. A too large architecture, i.e. with too many parameters (hidden nodes) may also lead to overfitting. What is likely to happen in these cases is that each pattern is simply stored exactly without the network learning the correlations within the examples. The network will therefore have little ability to generalize and is unlikely to give correct decisions or predictions when presented with the novel values of the test set (cf. Section 4.1.4.3).

Overfitting is especially dangerous because it can easily lead to predictions that are far beyond the range of the training data with many of the common types of artificial neural networks. But underfitting can also produce wild predictions in multilayer feedforward networks (cf. Section 4.5), even with noise-free data.

The best way to avoid overfitting, given ample training data, is to optimize the neural network architecture, i.e. by pruning irrelevant or redundant input units, hidden units and weights. Arbitrarily reducing the number of weights to compensate for lack of training data is risky, though, as it may in turn lead to underfitting.

Given a fixed amount of training data, there are a number of effective approaches to avoiding underfitting and overfitting, and hence getting good generalization. These include model selection, weight decay, early stopping and Bayesian estimation ([Fine, 1999], [Geman & Doursat, 1992], [Smith, 1993]), not all of which will be addressed in this thesis.

A summary of the various issues affecting generalization in artificial neural networks is given in Geman et al. [Geman & Doursat, 1992]. Chapter 5, dealing with one aspect of model selection, presents an algorithm for pruning excess hidden nodes. Sensitivity analysis, which deals specifically with pruning redundant input nodes, is also an active field of research ([Engelbrecht, 1999], [Engelbrecht & Fletcher, 1999]).

4.1.1.2 Data auditing

The data auditing should take place before any processing starts. Any problems that exist in the database (e.g. missing values, default values and unreliable and inconsistent data fields) will manifest themselves at the validation stage (see Section 4.1.4) if not dealt with now. One common and simple procedure is to generate histograms for the data (or subsets thereof) and scrutinize

them for completeness. It is usual to consider a data field as useful if at least 70% of the records contain values. Various techniques can be used to estimate these missing values, depending on the nature of the data and the preference of the user. For example, if the variable is an integer or a real number, the missing value can be replaced by the mean value for the field; if the data field is categorical, the missing value can be replaced with the mode for that category; simply flagging the missing values is another option.

4.1.2 Data preprocessing

One common misconception when developing artificial neural networks is that the raw data can be presented to the system, which will sort the useful input fields from the irrelevant ones to achieve a high performance network automatically. This is unfortunately not the case, and, as in Statistics, it is necessary to preprocess the data before presenting it to the system in order to develop a robust neural model.

The preprocessing involves a wide range of techniques for manipulating the data in order to extract the data fields that may be used by the artificial neural network. If the performance of the artificial neural network falls outside the required, predefined limits, the data preprocessing operations may have to be revisited.

The preprocessing techniques common to artificial neural network development are consistent with those used to develop classical statistical models ([Smith, 1993], [SPSS, 1995]), e.g.

- crosstabulations for categorical data or simple correlation analyses and scatter plots for continuous variables to identify input variables which do not contain any discriminatory or predictive information, as well as possible multicollinear variables (i.e. highly correlated input variables, containing similar information which may make no contribution to the

neural model; however, it should be borne in mind that artificial neural networks are designed specifically to take advantage of complex relationships between variables, therefore it may be necessary to train two networks: one with and one without the possibly multicollinear variable to assess an additional variable's contribution to a network model);

- histogram analyses to identify skewed variables - which should be transformed to attain a more symmetric distribution over the entire range of the input field - and outliers which should be dealt with to avoid hampering the performance of the neural model. (The input values are often normalized and outliers can distort this normalization, outliers should therefore be scrutinized for correctness; it appears to be common among users of artificial neural networks to "clip" variables with long tails by setting some limit to the range, however this should be done cautiously.)

4.1.3 Data encoding

The data (input and target patterns) in artificial neural network applications are usually of two broad types: quantitative variables or categorical variables. It must be ensured that the selected variables are encoded in a format compatible with the artificial neural network. This includes encoding categorical variables and normalizing continuous variables. In preparing the data for training the network, it must be decided how to represent the target outputs. It is in general easier for a network to learn a set of distinct responses than a continuous-valued response. However, artificially categorizing continuous target data must be done with caution as it may be more difficult for the network to learn examples that occur on, or near, the boundaries of the classes.

4.1.3.1 Input variables

4.1.3.1.1 Categorical variables All variables that are used to indicate which one of a (relatively) small list of possible distinct categories or attributes is observed for a sampling unit are classified as categorical variables, also sometimes called classification or qualitative variables. It is important that the categories of such a variable are distinct (i.e. clear-cut and well-defined), mutually exclusive (i.e. each sampling unit belongs to only one category) and exhaustive (i.e. there must be an appropriate category for each sampling unit).

Nominal variables Categorical variables where there is no particular ordering amongst the categories are known as nominal variables. In artificial neural network literature, nominal categorical variables are often referred to as class variables.

Binary variables, taking on the values 0 or 1, are common and are usually represented by only one node which indicates class membership. However, in many cases it may be advantageous to modify the network to accommodate inputs in bipolar form (-1 and 1) instead (cf. Section 4.3.3), as bipolar representation allows missing data to be represented by zero.

For multichotomous variables, simply assigning a numeric value to each category of a qualitative variable is not an effective coding strategy as it leads to unjustified linear relationships. One common technique to circumvent this problem is to use a form of indicator variable coding (also referred to as $1-\text{of}-N$ code in artificial neural network literature) where a string of N separate fields (i.e. a set of binary input nodes) each takes on the value 1 or 0 depending on the status of the variable, as displayed in Table 4.1 for a variable with four categories.

	Node			
	1	2	3	4
Category 1	1	0	0	0
Category 2	0	1	0	0
Category 3	0	0	1	0
Category 4	0	0	0	1

Table 4.1: 1−of−N coding

This encoding scheme differs from ordinary indicator variable coding for multiple categories where the required number of nodes is one less than the number of categories as either the first or the last category is the reference category.

The disadvantage of the 1−of−N method is that the number of inputs to the artificial neural network, and accordingly the number of weight parameters (cf. Section 4.4), is increased by the number of categories of the variable, hence increasing the time required to train and run the network. More importantly, more parameters in the network also require a larger sample size to achieve a given level of accuracy (cfSection 4.5.12). The problem can be partially overcome by merging categories that are logically similar, in relation to the problem at hand, within a variable.

Another problem associated with this method of data representation is that the network cannot generalize between classes. This is because the weights placed by the network on the connections between the nodes in the hidden layer and a particular input node representing one category have no effect on the network's output when that input is not turned on, i.e. each category is treated by the network as an independent variable [Smith, 1993].

Ordinal variables Ordinality refers to categorical variables where there is some natural ordering amongst the classes. One way to represent ordinal variables in artificial neural networks is by so-called thermometer coding [Smith, 1993]. A thermometer code is implemented by specifying a set of binary input nodes, each of which is either "on" or "off". The number of nodes is one less than the number of categories, e.g. for a four category ordinal variable, three nodes are needed: the first category is coded as all three nodes "on", the second category as the first two nodes "on", the third category as only the first node "on" and the last category as all three nodes "off". This is schematically represented in Table 4.2.

	Node		
	1	2	3
Category 1	1	1	1
Category 2	1	1	0
Category 3	1	0	0
Category 4	0	0	0

Table 4.2: Thermometer coding

Thermometer coding facilitates appropriate discrimination (each category is represented uniquely) and generalization (weights increase or decrease incrementally from one category to another) for an ordinal variable.

4.1.3.1.2 Continuous variables A variable is considered as continuous-valued when it takes on numerical values which can be any number within a range. Often variables with large variances contain more discriminatory information, but this is not a universal rule as it is dependent on the nature of

the specific data fields. Applying variables with different means and variances to a neural model will result in the fields with a wide range of values having larger weight values associated with them, and consequently having a greater effect on the response of the model. It is therefore not uncommon to normalize all inputs to ensure that they are used equally by the neural model. The recommended standard statistical method of subtracting the mean and dividing by the standard deviation of the variable is known as *zero mean unit standard deviation normalization* in artificial neural network jargon. Continuous-valued variables are usually represented by a single node in an artificial neural network.

4.1.3.2 Target variables

A single network may be designed to include more than one target variable. These variables may be either categorical or quantitative.

4.1.3.2.1 Categorical variables Very often the target variable of an artificial neural network is binary. This type of variable can be represented by two output nodes, each representing one of the two possible classes. It is, however, also possible to treat a binary target variable as quantitative, in which case there will be only one output node with a high value representing one outcome or class and a low value the other. Both approaches are workable, but using the latter has practical advantages as the computer algorithm can be designed to run faster; a single output node is also easier to interpret and use [Smith, 1993].

As discussed above (Section 4.1.3.1.1), bipolar representation may be more advantageous when encoding binary data as it allows missing data to be represented by zero. For target variables, this allows the network to distinguish between missing data and mistakes. Furthermore, if the aim is to generalize,

binary representation does not work as well as bipolar representation since the binary net never learns when the target is zero [Fausett, 1994].

When the target variable has more than two classes, the $1-of-N$ binary representation explained in Section 4.1.3.1.1, where each of the categories is represented by one output node, is used. The specific values chosen to represent "on" or "off" are not crucial to the network's performance, and often the numerical values 1 and 0 are assigned to these two categories respectively. As will become clear in Section 4.3.3, however, 1 or 0 are not values the output nodes can actually produce when an artificial neural network is trained. The output nodes can only approach these bounds. The more common practice is therefore to center the target values somewhat, with "on" for example represented by 0,9 and "off" by 0,1 [Smith, 1993]. These bounds correspond to the relatively linear portion of the sigmoid function, as displayed in Figure 4-1, which is commonly used as an activation function for the output nodes.

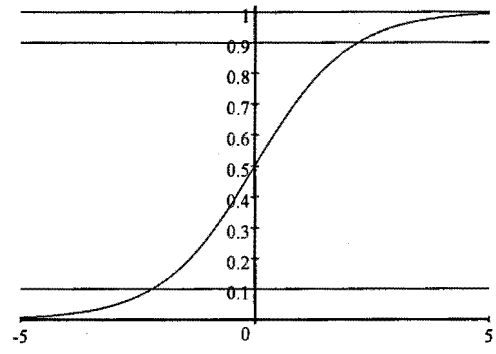

Figure 4-1: Near-linear range of the logistic sigmoid function

Even though the target outputs are binary, the actual outputs produced by the network are continuous-valued, taking on values between the bounds of the sigmoid function (when the sigmoid function is used as the activation function). The distribution of the actual output depends on the data and on

the degree to which categories overlap. Invariably the network output does not unambiguously indicate a single class for every pattern. It is common practice to use one of the following rules to decide which class a pattern belongs to, although there are no sound theoretical foundations for these decision rules [Smith, 1993]:

- Assign the pattern to the class whose output node has the highest network output value.
- Assign the pattern to the class whose output node has the highest network output value, provided that the value exceeds some predetermined minimum; otherwise remain undecided.
- Assign the pattern to the class whose output node has the highest network output value, provided that the value exceeds the next higher value by some predetermined minimum amount; otherwise remain undecided.

4.1.3.2.2 Continuous variables As is the case for input variables, only one node is needed to represent a continuous-valued target variable. It is necessary to scale the values of the variable to a range that is within the bounds of the output node's activation function (cf. Section 4.3). Again, it is common practice to center the target values to facilitate learning by the network.

Scaling the values of the target variable is simple:

$$T_j = T_{\min} + (\frac{V_j - V_{\min}}{V_{\max} - V_{\min}}(T_{\max} - T_{\min}))$$

where T_j denotes the j-th scaled target value, $T_{\max}$ (e.g. 0,9) and $T_{\min}$ (e.g. 0,1) the scaled maximum and minimum target values respectively, V_j the raw value of the variable and $V_{\max}$ and $V_{\min}$ the maximum and minimum raw values of the variable [Smith, 1993].

Interpreting the output produced by the artificial neural network model for a scaled continuous variable requires reversing this scaling:

$$R_j = V_{\min} + (\frac{O_j - T_{\min}}{T_{\max} - T_{\min}}(V_{\max} - V_{\min}))$$

where R_j is the rescaled network output and O_j the actual output of the network.

4.1.4 Data partitioning

Once a satisfactory data set that has been audited and preprocessed has been obtained, it must be partitioned into training, validation and test files.

4.1.4.1 Training files

Training files contain the data that are used to train the network. Since artificial neural networks are learning to associate the input from the training file with their corresponding targets, it is necessary to monitor and evaluate the artificial neural network's performance during training. This is usually measured by the error, i.e. the difference between the target values and the output values computed by the artificial neural network.

There are two sources of error. The first source is noise, a rather broad term which includes inaccuracies in the data introduced by such factors as inaccurate measuring instruments or the fact that the input variables do not contain all the information needed to determine the target variables. The second source of error is due to the mapping function's inability to fit the target function adequately.

If a network is left to train for too long, or if the architecture is too large, the model will have learned the characteristics of the training data almost perfectly. It is therefore inevitable that the neural model will learn the noise

characteristics of the training data given that the training data contain some noise. Noise is per definition unpredictable, and the network's performance on unseen data will be degraded by learning it. This phenomenon is termed overtraining (cf. Section 4.1.1.1), and is also common to traditional statistical models. With overtraining a network loses its ability to generalize.

4.1.4.2 Validation files

The problem of overtraining can be avoided by using a validation data set to monitor training, analogous to cross-validation in Statistics. The validation data consist of a small portion of the training data that are not used to build the neural model, but to monitor the performance of the neural system during the training process by measuring the error (between the network output and the target value) on the validation data at frequent intervals during the training cycle, and comparing it to the training error. This is best achieved by plotting the validation and training errors against the number of epochs. The training and validation errors drop at approximately the same rate during the early stages of learning. When the system begins to learn the noise characteristics of the training data, however, the validation error gradient will decrease and the error will eventually increase, indicating that training should stop [Hecht-Nielsen, 1990].

4.1.4.3 Test files

Once the artificial neural network model has been created, its suitability to the application must be investigated. This involves testing the performance of the neural system on unseen data. The test file contains data records that are kept aside and is therefore unknown to the trained network. During the so-called recognition phase the test data records are used to measure or confirm the expected performance of a trained application where the weights

are fixed. The test set is also sometimes referred to as the run set in artificial neural network applications. This is usually the case when there are no target outputs to predict.

4.1.4.4 Subset selection

The way in which the subsets are partitioned and selected is strongly influenced by the type of modelling task. Random selection of the training, validation and test records is recommended if the problem is a static prediction or classification task, as ideally, the training, validation and test files should all contain data that represent the entire range of the problem. Randomization will negate the effect of possible hidden correlations between records in the data, thus avoiding introducing dependencies in the model and obtaining biased estimates of the network's performance. This is especially important if the data have been collected in such a way that there is an ordering in the data (e.g. all the respondents in one area after another in a marketing problem). Randomizing time series data, however, will destroy the time history relationship inherent in the data. In this case special caution must be taken to preserve the time series. One way to handle this problem is to divide the time series into a number of equal blocks from which the training and validation and the test sets are then randomly selected. Usually the validation sets are taken from the end of each block.

The available amount of data determine how much data should be used for training and how much should be reserved for testing. One should aim to achieve a balance between the training (and validation) set and the test set, bearing in mind that a too small training set may lead to the problem of overfitting. On the other hand, even though using more data for training should result in a better artificial neural network model, there should still be sufficient data left to test whether this is indeed the case.

It is common practice to use a single subsample for both validation and testing. This method is fairly safe provided that the trained network is not noticeably affected by different ways of dividing the sample into training and validation subsamples. Smith proposes that two-thirds of the sample is used for training and one third for validation [Smith, 1993]. When it is necessary to construct separate validation and test files, the norm is 40% of the examples for training and 30% each for validation and testing.

4.2 Architecture of an artificial neural network

An artificial neural network is usually visualized as sets of neurons arranged in layers. The architecture of the network refers to the organization of the neurons in different layers with their corresponding pattern of connections. Neurons in the same layer behave similarly as they typically have the same activation function (cf. Section 4.3) and the same pattern of connections to other neurons in that layer (they may be either fully interconnected or not connected at all). Neurons from one layer are connected to neurons in the next layer. In general, artificial neural networks are classified into single-layer or multilayer networks. It is customary not to count the input layer when determining the number of layers since no computations are performed in this layer.

A number of excellent textbooks dealing extensively with artificial neural network architectures and algorithms have appeared in the last few years, amongst them [Zurada, 1992], [Smith, 1993], [Fausett, 1994], [Anderson, 1995] and [Haykin, 1999].

Since no common standards have yet been adopted in the technical literature, the notation will be introduced throughout this chapter as it is needed.

4.2.1 Single-layer artificial neural networks

A single-layer artificial neural network has only one layer of weights connecting the input and the output node layers. The following diagram illustrates a typical single-layer network with I input nodes (X_i), J output nodes (O_j) and weights w_{ji} connecting the j-th neuron (output node) with the i-th input node. It is common in the technical literature on artificial neural networks to use the double subscript for weights such that the first and second subscript denote the index of the destination and source nodes respectively. (The input and output vectors are invariably called input and output patterns in artificial neural network literature.)

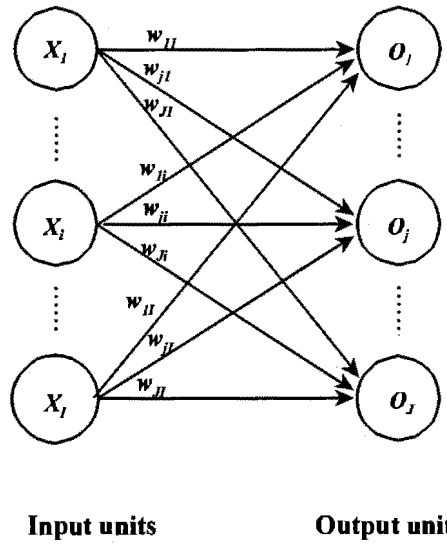

Figure 4-2: Single layer artificial neural network

Pattern classification is a typical example of where a single-layer artificial neural network is appropriate, i.e. when the classification of vectors in a single category is considered.

4.2.2 Multilayer artificial neural networks

Multilayer artificial neural networks have one or more layers of nodes (the so-called hidden nodes) between the input nodes and the output nodes. There is typically a layer of weights between each adjacent layer of nodes (input, hidden or output), resulting in at least two weight layers. Multilayer artificial neural networks are more powerful than single-layer networks as at least one hidden layer is required to perform nonlinear mappings.

4.2.2.1 Number of hidden layers

Although a many-layered topology is also valid, it has been formally shown that there is in practice seldom any need to go beyond a single hidden layer to obtain an arbitrarily accurate approximation to an arbitrary mapping, provided sufficiently many hidden nodes are available ([Cybenko, 1988], [Hornik & White, 1989], [Stinchcombe & White, 1989]). The functions computed by the artificial neural network that can approximate any function to any degree of accuracy are called universal approximators. Mathematical proofs of multilayer artificial neural networks as universal approximators have been provided by, amongst others, Cybenko and Hornik, Stinchcombe and White ([Cybenko, 1989], [Hornik & White, 1989], [Stinchcombe & White, 1989]).

4.2.2.2 Number of hidden nodes

The number of nodes in the system, and consequently the weights, should be directly related to the complexity of the system being modelled: the greater the number of hidden nodes available in the model, the more complex the function that the system can model. As discussed in Section 4.1.1.1 however, too many hidden nodes for the problem may result in the system becoming too specific or overtrained, hence a general solution will not be found. Each problem has its own optimum number of hidden nodes, and a degree of ex-

perimentation and experience is necessary when developing an artificial neural network. Numerous pruning algorithms have been developed to automatically remove irrelevant nodes, e.g. Optimal Brain Damage [LeCun & Solla, 1990] and the algorithm presented in Chapter 5.

4.2.2.3 Architecture

The architecture of multilayer artificial neural networks may be feedforward or recurrent (iterative). Figure 4-3 is an example of a multilayer artificial neural network with a feedforward configuration.

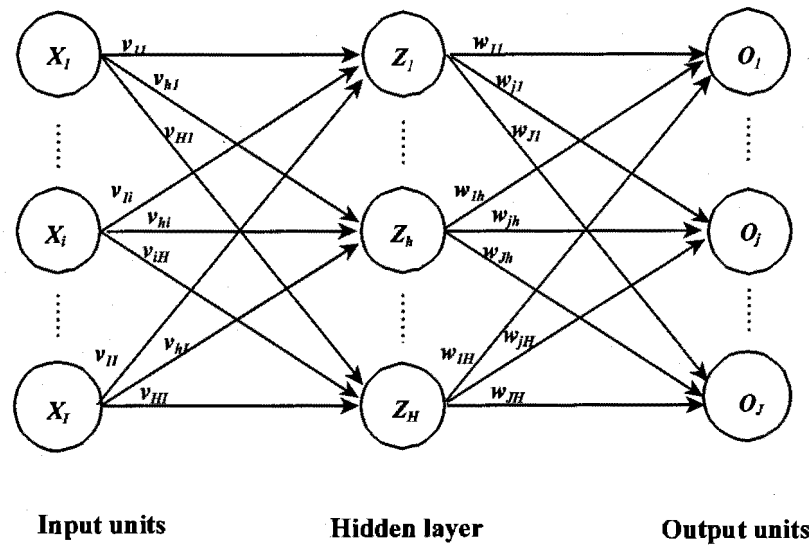

Figure 4-3: Multilayer artificial neural network

4.2.2.3.1 Feedforward artificial neural networks Feedforward artificial neural networks have no connections back to previous layers. Information flows from the I neurons in the input layer, possibly via the H nodes in the intermediate hidden layer to the J output nodes as illustrated in Figure 4-3. In this diagram weight v_{hi} connects the h-th neuron in the hidden layer with

the i-th input and weight w_{jh} connects the j-th neuron in the output layer with the h-th neuron in the hidden layer.

The convention for double subscripts differs amongst authors. Here the convention will be as explained in Section 4.2.1, namely the first and second subscript denote the index of the destination and source nodes respectively. Feedforward networks have no memory and recall is instantaneous. The network responds only to its present input.

4.2.2.3.2 Recurrent artificial neural networks Recurrent networks have feedback connections between different layers where connections among the nodes may even form closed loops, i.e. some or all the neurons are connected to themselves, allowing the output signals of neurons to be fed again to the inputs as illustrated in Figure 4-4.

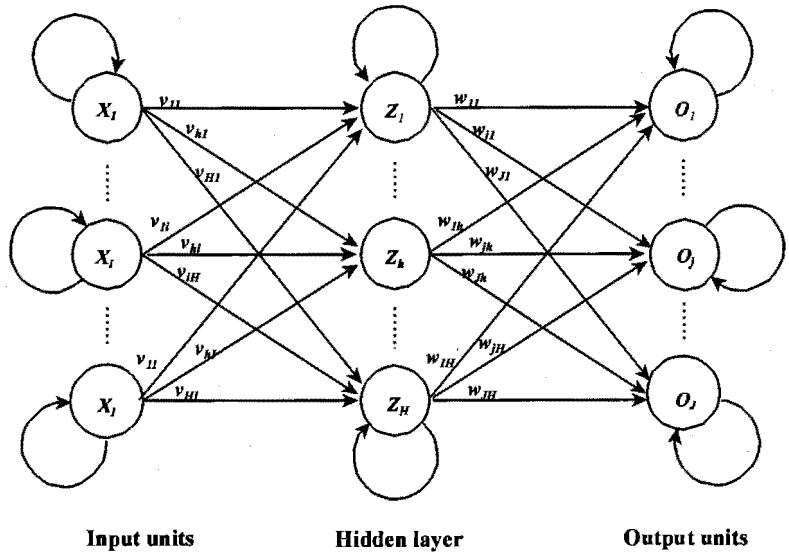

Figure 4-4: Recurrent artificial neural network

These networks can be considered as dynamical systems as they interact with their input through the output. A certain time interval is needed for their

recall to be completed. Recurrent networks are mainly used when there are temporal patterns in the data which is then learned by the neural network.

The configuration of recurrent artificial neural networks is especially suited to optimization problems ([Fausett, 1994], [Anderson, 1995]).

4.3 Activation functions

One of the functions of an artificial neuron is to sum its weighted input signals. This summation value is then evaluated using the threshold function, also known as the activation function or mapping function, to decide whether the node is activated or not. Typically, the same activation function is used for all neurons in the same layer. To maintain the advantages of multilayer artificial neural networks, as opposed to single-layer networks, nonlinear activation functions are commonly used. By using nonlinear and higher-order activation functions we are able to achieve nonlinear decision boundaries and more closely approximate any given output space region more tightly, often with fewer neurons, resulting in efficiency and invariably faster computations. The activation functions which are generally used are described next:

4.3.1 The identity function

The identity function $f(x) = x$ for all x, displayed in Figure 4-5, is generally used for the input nodes where no computation needs to be performed.

4.3.2 The binary step function

The binary step function (cf. Figure 4-6), or Heaviside function, is often used in single-layer networks to convert the total input to an output value that is a binary (1 or 0) or a bipolar (1 or -1) signal; θ is a fixed threshold value:

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases} .$$

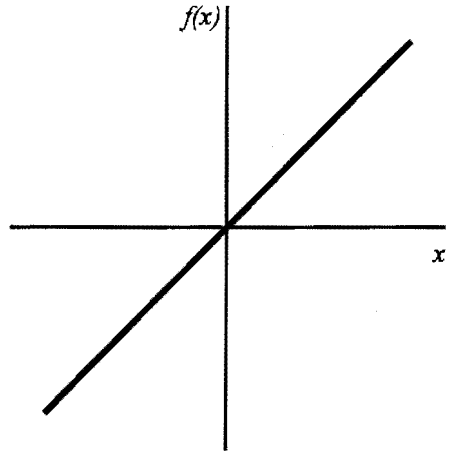

Figure 4-5: Identity function

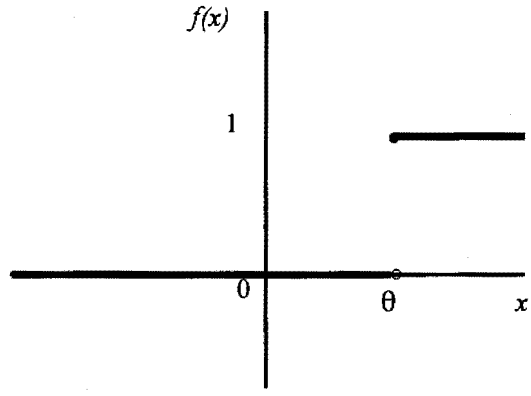

Figure 4-6: Binary step function

4.3.3 The sigmoid function

The continuous S-shaped curve obtained by the sigmoid function is particularly advantageous in artificial neural networks trained by backpropagation (cf. Section 4.5). This is because the simple relationship between the value of the function at a point and the value of the derivative at that point reduces the computations during training substantially [Fausett, 1994]. The hyperbolic tangent, binary sigmoid and bipolar sigmoid functions are most commonly used.

4.3.3.1 The binary sigmoid function

The range of the binary sigmoid or the logistic sigmoid function is (0;1). It is therefore an appropriate activation function for artificial neural networks where the desired output values are either binary or fall in the interval [0;1], although it is recommended to convert to bipolar form for binary data and use the bipolar sigmoid or hyperbolic tangent functions [Fausett, 1994].

The logistic sigmoid function is defined as

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

where σ represents the steepness, or slope, parameter.

Its first order derivative is

$$\begin{aligned} f'(x) &= \frac{\sigma \cdot e^{-\sigma x}}{(1 + e^{-\sigma x})^2} \\ &= \sigma f(x)[1 - f(x)]. \end{aligned}$$

Figure 4-7 illustrates the logistic sigmoid for four different values of the steepness parameter σ. It can be seen that the function becomes steeper as σ increases from 0,5 to 1, to 2 and then to 4.

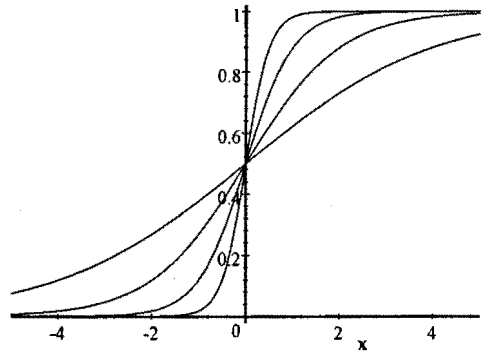

Figure 4-7: Logistic sigmoid function for different values of σ

4.3.3.2 The bipolar sigmoid function

The binary sigmoid function can be scaled to have any range of values that is appropriate for a given problem [Fausett, 1994]. For a given interval $[a; b]$, the sigmoid function

$$g(x) = \frac{b-a}{1+e^{-\sigma x}} + a$$

$$= (b-a)f(x) + a$$

will have a range of $(a; b)$, where $f(x)$ refers to the logistic sigmoid function.

The most common range is (-1;1), in which case the function is known as the bipolar sigmoid function.

It is defined as

$$\begin{aligned} g(x) &= 2\,f(x) - 1 \\ &= \frac{2}{1+e^{-\sigma x}} - 1 \\ &= \frac{1-e^{-\sigma x}}{1+e^{-\sigma x}} \end{aligned}$$

and its first order derivative is

$$\begin{aligned} g'(x) &= \frac{2\sigma e^{-\sigma x}}{(1+e^{-\sigma x})^2} \\ &= \frac{\sigma}{2}[1+g(x)][1-g(x)]. \end{aligned}$$

This function is displayed in Figure 4-8, again for steepness parameter values σ increasing from 0,5 to 1, to 2 and then to 4, illustrating that the function's slope increases accordingly.

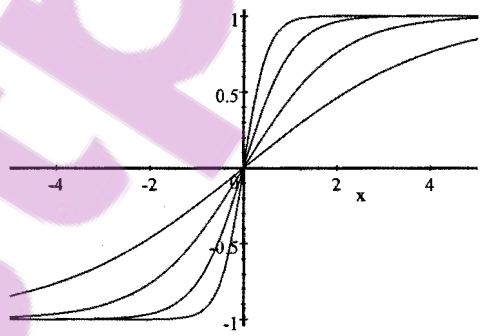

Figure 4-8: Bipolar sigmoid function for different values of σ

4.3.3.3 The hyperbolic tangent function

The bipolar sigmoid is closely related to the hyperbolic tangent function, which is also commonly used as the activation function when the target values' range is between -1 and +1. The hyperbolic tangent is

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$= \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

with first order derivative

$$h'(x) = [1 + h(x)][1 - h(x)].$$

The hyperbolic function is displayed in Figure ??.

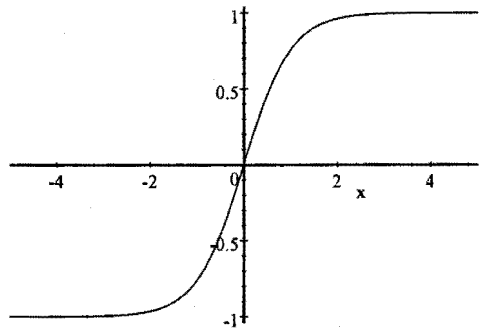

Figure 4-9: Hyperbolic tangent function

4.4 Training algorithms

Artificial neural networks differ from each other in their learning modes. Learning corresponds to parameter changes, i.e. the adjustment of the weights connecting the nodes. The training or learning algorithm specifies the setting and adjusting of weights on the connections in the artificial neural network in such a way that the network acquires the desired behaviour. This is done once the artificial neural network architecture has been decided upon.

Three broad classes of artificial neural network models can be identified, based on the type of training: self-supervised, supervised and unsupervised artificial neural networks. The kind of problem that needs to be solved will determine the appropriate type of training and therefore the appropriate artificial neural network model to be implemented.

A number of different artificial neural network learning rules which are commonly used are briefly outlined in this Section.

4.4.1 Self-supervised training

No learning takes place in artificial neural networks with fixed weights. These nets are known as self-supervised. This type of training method is suitable to constrained optimization problems (i.e. where not all constraints can be satisfied simultaneously). In the design of this type of artificial neural network the weights are set to represent the constraints and the quantity to be maximized or minimized. Self-supervised neural nets are also referred to as fixed-weight nets or batch-learning ([Zurada, 1992], [Fausett, 1994]).

An example of a fixed-weight algorithm is the artificial neuron specified by McCulloch and Pitts ([McCulloch & Pitts, 1943], discussed in Chapter 3).

4.4.1.1 Hebb learning rule

The earliest and simplest rule for an artificial neural network was formulated by D. O. Hebb ([Hebb, 1949], cf. Chapter 3). The Hebb rule is an extension of Hebb's original proposal that learning occurs by increasing the weights between two interconnected neurons that are activated at the same time, strengthening the output in turn for each positively correlated input that it is presented with. The extended Hebb rule, which also adapts the weights if both neurons do not fire at the same time, facilitates a stronger form of learning as it allows the net to decrease weights [McClelland & Rumelhart, 1988]. A Hebb net is a single-layer, feedforward, unsupervised artificial neural network trained with the (extended) Hebb rule.

Weights are initialized to small random values around 0, and weight updates, i.e. learning, are performed using the rule

$$w_i(\text{new}) = w_i(\text{old}) + tx_i$$

where w_i is the weight associated with input x_i, $i = 1, 2, \cdots, I$ and t is the corresponding target (observed) output.

The network was developed for binary data representation (0 and 1) but can in most cases be modified to accommodate inputs and outputs in bipolar form (-1 and 1). Bipolar representation allows missing data to be represented by zero, thus allowing the net to distinguish between missing data and mistakes. If the aim is to generalize, binary representation does not work as well as bipolar representation since the binary net never learns when the target is zero.

4.4.2 Supervised training algorithms

Most commonly, artificial neural networks use supervised training which require both inputs (a set of training patterns or vectors) and their associated outputs (target vectors). The actual output of the network may or may not match the target output, depending on the weights at that particular moment. The training algorithm modifies the network weights so that the model learns the mapping from the inputs to the desired target.

The problems suitable to supervised learning are:

- Classification problems where the neural model assigns examples into one of I groups.
- Pattern recognition or association.
- Time series forecasting where time ordered information is used for prediction.
- Prediction problems where the neural model assigns a probability of occurrence to unseen data.

4.4.2.1 Perceptron

The perceptron learning rule which was formulated by Frank Rosenblatt is a more powerful learning rule than the Hebb rule to classify inputs as belonging, or not belonging, to a particular class ([Rosenblatt, 1958], [Rosenblatt, 1962], cf. Chapter 3). In this single layer neural net the response of the output node for each training input is calculated. It is then determined whether an error occurred for this pattern by comparing the calculated output with the target value, thus using supervised learning to adjust the artificial neural network weights.

The perceptron learning rule convergence theorem states that, under suitable conditions, its iterative learning procedure will converge to the correct weights in a finite number of steps. The correct weights are those that will allow the network to give the correct output for each of the input training patterns, i.e. the net will learn the classification. One of the conditions is that weights exist that will allow the net to respond correctly to all training patterns.

Originally, the perceptron used binary inputs with a bipolar target, although it is shown in Fausett [Fausett, 1994] that the algorithm is also suitable for bipolar input vectors, as well as for the case where the input vectors belong to one (or more) of several output classes. The output is $o = f(o_{in})$, where $o_{in} = b + \sum_i x_i w_i$ with $\mathbf{x}$ the I-tuple input vector with its corresponding I-tuple weight vector $\mathbf{w}$, and b an adjustable bias. Not including a bias in the model will result in the network finding a decision boundary for classification that is forced to go through the origin. In many cases this may change a problem that could be solved (i.e. one for which weights for a separating line or plane exist) into a problem that could not be solved [Fausett, 1994].

The activation function of the perceptron is the binary step function with an arbitrary, fixed, non-negative threshold θ:

$$f(o_{in}) = \begin{cases} 1 & \text{if } o_{in} > \theta \\ 0 & \text{if } -\theta \leq o_{in} \leq \theta \\ -1 & \text{if } o_{in} < -\theta \end{cases} .$$

For this activation function there are actually two thresholds, θ and -θ, which define two decision boundaries and three output spaces: 0, 1 and -1. The region of positive response from that of negative response is separated by an "undecided" fixed width band, determined by the value of θ, corresponding to 0.

The weights (and bias) are adjusted only when an error occurs, that is, when the network output o differs from the actual target output t. As more training patterns produce the correct response, less learning occurs, since by definition weights are adjusted only when an error occurs. Training stops when there are no more weight changes. Weights are adjusted according to the formula

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$

where α is the learning rate[1] α, $0 < \alpha \leq 1$. The algorithm is not particularly sensitive to either the initial values of the weights or the value of the learning rate. The initial weights are often set to 0 and α to 1.

4.4.2.2 ADALINE

The ADALINE (ADAptive LInear NEuron), developed by Bernard Widrow and Marcian Hoff, is very similar to a perceptron ([Widrow & Hoff, 1960], cf. Chapter 3). It is a single neuron that typically uses bipolar activations for its input signals and its target output, although it is not restricted to these values.

As for the perceptron, the activation of the node is its net input, i.e. the identity function $o_{in} = b + \sum_i x_i w_i$, $i = 1, 2, \cdots, I$. For binary or bipolar target values a step function can be used as the activation function for the ADALINE's output node $O = f(o_{in})$:

$$f(o_{in}) = \begin{cases} 1 & \text{if } o_{in} \geq 0 \\ -1 & \text{if } o_{in} < 0 \end{cases}.$$

[1]The learning rate is a parameter that controls the amount by which weights are changed during training. In some nets the learning rate is a constant; in others it is reduced as training progresses to achieve stability [Fausett, 1994].

An ADALINE also uses supervised training to adjust the network's weights. The network is trained using the Widrow-Hoff learning rule, also know as the least mean squares (LMS) rule as the learning rule minimizes the mean squared error between the network output and the target value over all training patterns. This is accomplished by reducing the error for each pattern, one at a time.

Like the perceptron, the Widrow-Hoff rule will converge if appropriate weights exist. The rule allows the net to continue learning on all training patterns, even after the correct output value is generated. This is in contrast with the perceptron where less learning occurs as more training patterns produce the correct response.

The weights (including the bias) are updated with the formulae

$$b(\text{new}) = b(\text{old}) + \alpha(t - o)$$

$$w_i(\text{new}) = w_i(\text{old}) + \alpha(t - o)x_i.$$

The notation is again as above where $\mathbf{x}$ is the I-tuple input vector with its corresponding I-tuple weight vector $\mathbf{w}$ and b an adjustable bias. The differences between the perceptron and an ADALINE are that the threshold function is slightly different, and more importantly, that the weight update is proportional to the difference between the target output and actual output, rather than simply proportional to the target output itself. Training will continue over the given set typically until a specified error tolerance has been met or until a preset number of training epochs (full cycles through the training set) has been reached.

Small random values are usually used to initialize the weights as the algorithm is not sensitive to the initial weights. Setting the learning rate α,

however, requires some care. If the value chosen is too large, the learning process will not converge; with a too small value, learning will be extremely slow. A practical range for the learning rate for a single neuron is proposed in Fausett as $0,1 \leq \alpha \leq 1$ [Fausett, 1994].

The Widrow-Hoff or least mean squares rule can be extended to allow for more than one output, in which case the weights are changed to reduce the difference between the net input to the output node, o_{in_j}, and the target value t_j, where $j = 1, 2, \cdots, J$ for the J outputs. This formulation reduces the error for each pattern. The $I \times J$ weight matrix has elements w_{ji}, sticking to the convention that the first and second subscript denote the index of the destination and source nodes respectively, i.e. w_{ji} denotes the weight on the connection between the j-th output node and the i-th input node.

In this case

$$o_{in-j} = \sum_{i=1}^{I} x_i w_{ji}$$

for the J-tuple computed output vector for the I-tuple input vector $\mathbf{x}$.

The weight updates for the connection between the j-th output neuron and the i-th input node

$$w_{ji}(\text{new}) = w_{ji}(\text{old}) + \alpha(t_j - o_{in-j})x_i$$

are usually expressed in terms of the weight change as

$$\Delta w_{ji} = \alpha(t_j - o_{in-j})x.$$

A MADALINE (Many ADAptive LInear NEurons) is an extension of the ADALINE algorithm to multiple layers in order to accommodate several output nodes.

4.4.2.3 Delta rule

The delta learning rule was introduced by McClelland and Rumelhart ([McClelland & Rumelhart, 1986], [McClelland & Rumelhart, 1988], cf. Section 3) for networks in the supervised training mode. Mappings in which the input patterns are linearly independent can be solved using a single-layer net with this rule. It is really an extension of the Widrow-Hoff or least mean squares rule where the modification allows for a continuous, differentiable and monotonically nondecreasing activation function to be applied to the output nodes of the network [Zurada, 1992]. Differentiability is needed because the derivative of the activation function is used to compute the weight updates for this gradient descent method to minimize the total squared error of the output computed by the net.

The learning rule is easily derived from the condition of least squared error between the network output $o_j = f(o_{in-j})$ and the target t_j, $j = 1, 2, \cdots, J$ where

$$\begin{aligned} o_{in-j} &= \sum_{i=1}^{I} x_i w_{ji} \\ &= \mathbf{w}_j^t \mathbf{x}. \end{aligned}$$

One can therefore also write $o_j = f(\mathbf{w}_j^t \mathbf{x})$, $j = 1, 2, \cdots, J$.

The least squared error is defined here as

$$E \triangleq \frac{1}{2} \sum_{j=1}^{J} (t_j - o_j)^2$$

or

$$E = \frac{1}{2} \sum_{j=1}^{J} (t_j - f(\mathbf{w}_j^t \mathbf{x}))^2.$$

The error gradient vector value is

$$\nabla E = -\sum_{j=1}^{J}(t_j - o_j)f'(\mathbf{w}_j^t\mathbf{x})\mathbf{x}.$$

For the arbitrary weight w_{ji} the error gradient component is

$$\frac{\partial E}{\partial w_{ji}} = -\sum_{j=1}^{J}(t_j - o_j)f'(\mathbf{w}_j^t\mathbf{x})x_i \qquad \text{for } i = 1, 2, \cdots, I.$$

Noting that the weight w_{ji} only influences the error at output node o_j, the equation reduces to

$$\frac{\partial E}{\partial w_{ji}} = -(t_j - o_j)f'(\mathbf{w}_j^t\mathbf{x})x_i \qquad \text{for } i = 1, 2, \cdots, I.$$

The mathematical basis for this rule is the optimization technique known as gradient descent (or hill-climbing). The gradient of a function (here the squared error E) gives the direction in which the function increases more rapidly. The minimization of the error therefore requires the weight changes to be in the negative gradient direction. Incorporating the learning rate α, $0 < \alpha \leq 1$, the weight changes for the most rapid reduction in the local error is obtained as

$$\Delta\mathbf{w}_j = \alpha(t_j - o_j)f'(o_{in-j})\mathbf{x},$$

or, for the individual weight adjustments,

$$\Delta w_{ji} = \alpha(t_j - o_j)f'(o_{in-j})x_i \qquad \text{for } i = 1, 2, \cdots, I.$$

As for the ADALINE, the weights are initialized at any value for this method of training.

4.4.3 Unsupervised training algorithms

Unsupervised artificial neural networks extract correlations inherent in the data to discover patterns or regularities and relationships between the different parts of the input, and therefore do not need targets. The artificial neural network model is presented with a sequence of input vectors which then modify the weights so that the most similar input vectors are assigned to the same output node (referred to as a cluster unit in [Fausett, 1994]). The changes in the network parameters in discovering the existence of patterns, regularities and separating properties is called self-organization ([Fausett, 1994], [Haykin, 1999]). Zurada shows that learning is not necessarily possible in an unsupervised environment [Zurada, 1992].

These models are appropriate when the problem requires data segmentation. Another important function that can be performed by unsupervised artificial neural networks is feature detection, which is usually related to the dimensionality reduction of data. (The input vectors, or variables, are often called features in artificial neural network applications.) One important application involves mapping speech features using word phonemes to produce phonotonic maps which could enable profoundly deaf people to receive visual feedback from their speech ([Kohonen, 1988], [Kohonen, 1990]).

4.4.3.1 Kohonen self-organizing maps

The most popular unsupervised network is the Kohonen network, named for its Finnish inventor Professor Teuvo Kohonen ([Kohonen, 1977], [Kohonen, 1984], cf. Chapter 3). Kohonen nets do not associate an input pattern with a target output. Instead, the input signal $\mathbf{x}$ is considered an I-tuple which is subsequently classified as belonging to one of J cluster units. This self-organizing artificial neural network is also called a topology-preserving map as it assumes a topological structure among the cluster units.

In Kohonen learning, the nodes that update their weights do so by forming a new weight vector that is a linear combination of the old weight vector and the current input vector. As Fausett describes the self-organization process, the cluster unit whose weight vector matches the input pattern most closely is chosen as the winner [Fausett, 1994]. Typically the square of the minimum Euclidean distance between the input vector and the weight vector is used to determine the winning weight vector, although the scalar product of the input and the weight vector, simply giving the net input to the cluster unit, is also commonly used. If both units are standardized to unit length, the largest scalar product corresponds to the smallest angle between the input and weight vectors. The scalar product may also be interpreted as the correlation between the input and weight vectors. The testing function to determine the winner is given by the minimum $D(j)$ such that

$$D(j) = \sum_i (w_{ji} - x_i)^2.$$

Thus, the connection weights serve as a cluster exemplar of the input patterns associated with that cluster instead of an input scaling function. Only the winning unit and neighboring units (in terms of the topology of the cluster units, not in terms of weight vector similarity) update their weights by

$$\begin{aligned} w_{ji}(\text{new}) &= w_{ji}(\text{old}) + \alpha(x_i - w_{ji}(\text{old})) \\ &= \alpha x_i + (1-\alpha) w_{ji}(\text{old}). \end{aligned}$$

This weight update occurs for all i weights of every output unit j within a specified neighbourhood of radius R around the winning cluster W. Generally, both R and α decrease as learning progresses.

Weights are initialized using random values unless some information concerning the distribution of clusters is available, in which case the initial weights

can be taken to reflect the prior knowledge.

Kohonen nets are similar to a variety of competitive nets. The defining characteristic of these nets is that they choose one or more output neurons that will respond to any given input pattern, instead of providing an output pattern using all J output neurons. They can therefore be viewed as a nonlinear extension of principle components analysis.

Other types of artificial neural networks based on competition, most of which use Kohonen learning, are Maxnet, Mexican Hat, Hamming and Learning Vector Quantization (LVQ). These nets are discussed in more detail in several textbooks, e.g. Zurada and Fausett ([Zurada, 1992], [Fausett, 1994]).

4.4.3.2 Adaptive resonance theory

Adaptive resonance theory (ART) nets were designed by Carpenter and Grossberg to allow the user to control the degree of relative similarity of input patterns assigned to the same cluster ([Grossberg, 1977], [Fausett, 1994], cf. Chapter 3). The input vectors are clustered using unsupervised learning. The net has three layers of neurons; an input, an interface and an output cluster layer. There is both a forward and a backward connection between each interface and cluster neuron. The forward connection weights to the output cluster layer determine the winning cluster as the cluster with the largest net input upon presentation of an input pattern. The backward connection from the cluster layer to the interface layer determines whether the input pattern is similar to that cluster's exemplar vector. If so, only that cluster is allowed to update its weights; if not, the cluster is rejected and a new winner is chosen by repeating the algorithm.

The net thus resonates as learning occurs. Different clusters may be chosen for the same input pattern depending on when it is presented. A stable net will not oscillate among different cluster units during training. A plastic net is able

to respond to a new pattern equally well at any stage of learning. Adaptive resonance theory nets are designed to be both stable and plastic.

4.5 Backpropagation of error in a multilayer feedforward artificial neural network

The learning procedures discussed for supervised feedforward networks in Section 4.4.2 are applicable to single layer networks that are suitable for classification with linearly separable input patterns. The networks use a linear combination of inputs and weights with the weights as proportional coefficients. The argument of the nonlinear component, the activation function, is simply computed as the scalar product of the weight and input vectors. However, to train patterns that are not linearly separable, it is necessary to introduce a multilayer network consisting of the input layer, at least one hidden layer and the output layer. (As mentioned in Section 4.2.2, although in some cases a slight advantage may be realized by using two hidden layers, in general a single hidden layer is sufficient [Cybenko, 1988], [Hornik & White, 1989].) Feedforward artificial neural networks with one or more layers of nodes between the input and output nodes are also known as multilayer perceptrons in artificial neural network literature.

Multilayer artificial neural networks have been known for a long time, but the lack of appropriate training algorithms has prevented their successful applications for practical tasks. The delta training rule, introduced by McClelland and Rumelhart (cf. Section 4.4.2.3), cleared this obstacle.

The training method, called the backpropagation of error, uses the delta training rule and is the most popular algorithm for adjusting weights during the training phase of a feedforward artificial neural network. As discussed previously, it is simply a gradient descent method to minimize the total squared

error of the output computed by the net. This method is often also referred to as steepest descent. The very general nature of this method means that multilayer, feedforward artificial neural networks trained by backpropagation can be used to solve problems in virtually every field that uses supervised neural nets, i.e. for problems that involve mapping a given set of inputs to a specified set of target outputs. The three layer backpropagation network has become the industry standard.

Figure 4-10 displays a standard multilayer backpropagation artificial neural network with one hidden layer (the Z nodes). Both the output nodes (the O nodes) and the hidden nodes may include a bias node as shown. These bias terms act like weights on connections from nodes whose output is set to 1. Information flows from the I neurons in the input layer via the intermediate hidden layer with H hidden nodes to the J output nodes. Again, the subscript convention is such that v_{hi} denotes the weight that connects the h-th neuron in the hidden layer with the i-th input node and weight w_{jh} connects the j-th neuron in the output layer with the h-th neuron in the hidden layer. The bias on a typical hidden node Z_h is denoted by v_{h0} and the bias on a typical output node O_j is denoted by w_{j0}. Very often the bias nodes are not displayed explicitly in an artificial neural network diagram.

4.5.1 The backpropagation algorithm

The training of a network using the backpropagation algorithm involves three stages: the feedforward of the input training patterns, the calculation and backpropagation of the associated error and finally the adjustment of the weights. The algorithm was first described by Paul Werbos in his PhD thesis [Werbos, 1974] and is explained in numerous textbooks (e.g. [Zurada, 1992], [Smith, 1993], [Fausett, 1994], [Anderson, 1995], [Bishop, 1995], [Vapnik, 1995], [Fine, 1999], [Haykin, 1999]).

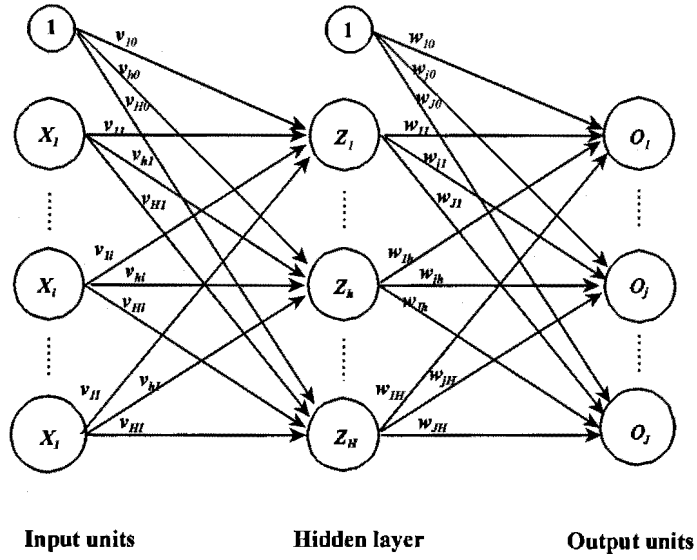

Figure 4-10: Multilayer feedforward artificial neural network

During the feedforward stage of the input training patterns each input node $X_i, i = 1, 2, \cdots, I$, is transmitted to each of the hidden nodes $Z_1, Z_2, \cdots, Z_H$ in the following layer, which in turn propagates the activations z_h obtained in each node from the input layer to each output node O_j, $j = 1, 2, \cdots, J$. The response of the net for the given input pattern is obtained by the computed activation o_j.

Analogous to the previous algorithms, the output signal, or activation, of node Z_h is $z_h = f(z_{in-h})$ for $h = 1, 2, \cdots, H$ where

$$z_{in-h} = v_{h0} + \sum_{i=1}^{I} x_i v_{hi}.$$

The output of node O_j is $o_j = f(o_{in-j})$ where

$$o_{in-j} = w_{j0} + \sum_{h=1}^{H} z_h w_{jh}.$$

The activation function f (cf. Section 4.3) should be continuous, differentiable and monotonically nondecreasing. Differentiability is needed because the derivative of the activation function is used to compute the weight updates (backpropagate the error). For computational efficiency, it is therefore advantageous if the activation function's derivative is easy to compute. Both the binary and bipolar sigmoid functions are commonly used as activation functions (cf. Section 4.3.3). The form of the target values is an important factor in choosing the appropriate activation function.

The training stage of the backpropagation algorithm involves the calculation and backpropagation of the error associated with a specific pattern. For each output node the computed activation o_j is compared with its corresponding target value t_j to determine the difference, namely the associated error for that pattern with that specific node, which is then propagated back into the net in the reverse direction ($j = 1, 2, \ldots, J$).

A factor δ_j, reflecting the portion of error weight adjustments for w_{jh} that is due to an error at output O_j, is then computed for each output node:

$$\delta_j = (t_j - o_j) f'(o_{in-j}).$$

δ_j thus contains the information about the error at node O_j.

Also calculated is the weight correction term for output O_j

$$\Delta w_{jh} = \alpha \delta_j z_h$$

and its bias correction term

$$\Delta w_{j0} = \alpha \delta_j.$$

As before, α is the learning rate, $0 < \alpha \leq 1$. These correction terms are

used in the final step of the algorithm to update the weights.

The error information term is δ_j is also used to distribute the error at output node O_j back to all nodes in the hidden layer below that feed into O_j. This is achieved by summing the delta inputs for each hidden node Z_h, $h = 1, 2, \cdots, H$, from the nodes in the layer above:

$$\delta_{in-h} = \sum_{j=1}^{J} \delta_j w_{jh}.$$

An error information term δ_h, reflecting the portion of error correction weight adjustment for v_{hi} that is due to the backpropagation of error information from the output layer to the hidden node Z_{hi} is computed next:

$$\delta_h = \delta_{in-h} f'(z_{in-h}).$$

Also computed is a weight correction term Δv_{hi}, which will be used to update the weights v_{hi} at the next stage,

$$\Delta v_{hi} = \alpha \delta_h x_i$$

as well as a bias correction term Δv_{h0},

$$\Delta v_{h0} = \alpha \delta_h.$$

This is done for each hidden node Z_h, $h = 1, 2, \cdots, H$. It is not necessary to propagate the error at output node O_j all the way back to the input layer as only the weights between the hidden and the input layer are adjusted.

In the final step of the backpropagation process, all the weight and bias updates for all layers calculated in the previous stage are adjusted simultaneously by adding the weight and bias corrections to their respective nodes to achieve

the new weights and biases. These updates occur for all neurons in both the hidden and the output layer, i.e. each output node O_j, $j = 1, 2, \cdots, J$, updates its bias and weights

$$w_{jh}(\text{new}) = w_{jh}(\text{old}) + \Delta w_{jh} \qquad \text{for } h = 0, 1, 2, \cdots, H$$

and each hidden node Z_h, $h = 1, 2, \cdots, H$, updates its bias and weights

$$v_{hi}(\text{new}) = v_{hi}(\text{old}) + \Delta v_{hi} \qquad \text{for } i = 0, 1, 2, \cdots, I.$$

This stochastic or online training proceeds iteratively with error corrections in the final step of each pattern presentation to adjust weights until a preset stopping condition (usually meeting a specified error tolerance or reaching a preset number of training epochs) is satisfied. Typically, a backpropagation artificial neural network needs many epochs for training. Various heuristics for improving the rate of convergence have been proposed, some of which are outlined in the sections below.

4.5.2 Training errors

For the purpose of weight adjustment in a single training step, the error to be reduced is computed for a pattern currently applied at the input of the network. Input patterns are submitted sequentially during the backpropagation training. However, for the purpose of assessing the performance of training, the joint error for the entire set of training patterns need to be computed, i.e. for an entire epoch.

The error expression to be minimized for a specific pattern p includes all squared error at the outputs o_j, $j = 1, 2, \ldots, J$:

$$E_p = \frac{1}{2}\sum_{j=1}^{J}(t_{pj} - o_{pj})^2, \qquad p = 1, 2, \ldots, P.$$

The joint error which is needed to assess the specified error tolerance is computed over the backpropagation training cycle:

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{J}(t_{pj} - o_{pj})^2.$$

Another error measure that is less influenced by the number of patterns (P) and the number of output nodes (I) is the root-mean-square normalized error [Zurada, 1992]:

$$E_{rms} = \frac{1}{PJ}\sqrt{\sum_{p=1}^{P}\sum_{j=1}^{J}(t_{pj} - o_{pj})^2}.$$

4.5.3 Derivation of the learning rule

Analogous to the derivation of the delta weights in Section 4.4.2.3, it can easily be shown, by applying the chain rule, that the gradient of the error function of a specific pattern p for an arbitrary weight w_{jh} between the output and the hidden layers is derived as

$$\frac{\partial E}{\partial w_{jh}} = -(t_j - o_j)f'(o_{in-j})z_h \qquad \text{for } j = 1, 2, \cdots, J.$$

The subscript p in the error term E_p, denoting that the error gradient is derived for a specific pattern p, has been dropped for brevity.

Incorporating the learning rate $\alpha, 0 < \alpha \leq 1$, and considering that the minimization of the error requires the weight changes to be in the negative

gradient direction, one obtains the individual weight adjustments for the most rapid reduction in the local error as

$$\begin{aligned} \Delta w_{jh} &= \alpha(t_j - o_j) f'(o_{in-j}) z_h \\ &= \alpha \delta_j z_h \end{aligned}$$

using the notation for the error information term between the output and hidden layers introduced above.

Similarly, for the arbitrary weight v_{hi} between the hidden and the input layers, the error gradient component is

$$\frac{\partial E}{\partial v_{hi}} = -\sum_{j=1}^{J} \delta_j w_{jh} f'(z_{in-h}) x_i \qquad \text{for } i = 1, 2, \cdots, I.$$

The weight changes for the most rapid reduction in the local error in this case is

$$\begin{aligned} \Delta v_{hi} &= -\alpha \frac{\partial E}{\partial v_{hi}} \\ &= \alpha \sum_{j=1}^{J} \delta_j w_{jh} f'(z_{in-h}) x_i \\ &= \alpha \delta_h x_i \end{aligned}$$

again using the notation for the error signal, this time between the hidden and input layers, introduced above.

The weight updates are thus indeed performed in the direction in which the function (i.e. the squared error between the target and output nodes) decreases most rapidly (hence steepest descent).

4.5.4 Initial weights and bias choices

In setting up the network it is important to pay attention to the initial weight settings when using backpropagation. The usual approach of minimizing the squared errors leads to a system of linear equations with a solution that is not necessarily unique. The choice of the initial weights will to some extent influence whether the network reaches a global, or only a local, minimum of the error function, as well as how quickly it converges to this minimum.

The update of the weights between any two neurons depends on both the upper and lower neurons' activation functions as well as on the derivatives of the activation functions of the neurons. It is therefore important to avoid choosing initial weights that are likely to result in derivatives of activations that are close to zero as the weight changes will then be very small and the network learning will be extremely slow. Too large values for the initial weights may produce initial output signals to each hidden or output node in the saturation region of the sigmoid function where the derivative has a very small value.

The aim is to begin with weight settings that result in a weighted sum of inputs close to zero for every node. This will result in output nodes with values close to the midpoint of the sigmoid function i.e. close to 0,5 for the binary logistic function and close to zero for the bipolar sigmoid function (cf. Section 4.3.3). Having activations in the midrange of the sigmoid activation function, where the gradient is steepest, will result in derivatives in a range that have proportionally larger contributions to weight changes than in the tail areas where the slope diminishes rapidly (cf. Section 4.5.6).

Smith mentions two ways of arranging the initial weights such that the weighted sum of inputs are close to zero for every node, hence producing midrange outputs [Smith, 1993]. The first method is appropriate for hidden nodes when the logistic sigmoid activation function is used. By setting the initial weights v_{hi} close to zero, the hidden node activations z_h will have midrange

values regardless of the values of its inputs. Another way to determine the magnitude of the initial weights between the input and hidden nodes is to consider the magnitude of the input values. For example, to ensure that the activation $z_h = f(z_{in-h})$ of hidden node Z_h is approximately between $0,25$ and $0,75$, the weighted sum of the inputs z_{in-h} must be somewhere between -1 and 1 for the logistic sigmoid activation function, and between $0,5$ and 2 for the bipolar sigmoid activation function.

Care should be taken that the all initial weights from the different hidden nodes on the same input are not equal. If they are, then all the hidden nodes will see the same input on every example, compute the same output and consequently make the same contribution to the network's error. The error derivatives with respect to these weights will consequently all be the same, with the result that all the weights will be changed by the same amount, and will remain the same regardless of how long the network is trained or how fast the network learns.

The output nodes O_j should also start with weighted sums of inputs o_{in-j} close to zero to attain outputs o_j within the midrange values of the activation function. However, setting the weights w_{jh} close to zero, as for the hidden nodes' weights, is in this case counterproductive. This is because the output node weights w_{jh} are used when computing the error derivatives of the hidden node weights in backpropagation. Small output node weights will result in small derivatives for the hidden nodes weights and accordingly small changes in the weights w_{jh}. The hidden nodes will only begin to learn rapidly when the weight connections to the output nodes become large enough for their contribution to be significant. However, the weight connections will remain small until the hidden nodes find something useful to do. This is a "Catch 22" situation which can only be resolved by initializing the weights w_{jh} with values larger than zero. The second method described by Smith to restrict the

activations of the output nodes to the midrange of the activation function is to initialize half the weights with say 1, in which case the other half of the weights should be initialized with -1. (If there are an odd number of weights the bias weight can be initialized with zero.) To obtain weighted sums o_{in-j} that are close to zero, it is necessary that the hidden nodes' activations are approximately equal (in the middle of the sigmoid function's range as described above) as the sum of similar hidden node outputs z_h multiplied by weights w_{jh} that are equal in absolute value, will be about zero.

4.5.4.1 Random initialization

It is commonly acceptable to randomly assign initial values to the weights and biases. The sign of the weights are immaterial as the final weights after training may be either positive or negative. It is important to train the artificial neural network with different sets of random weights to obtain an optimum solution.

4.5.4.2 Nguyen-Widrow initialization

D. Nguyen and B. Widrow made a simple modification to the random weight initialization to give much faster learning [Nguyen & Widrow, 1990]. This is accomplished by introducing a scale factor β that is a function of the number of input nodes, I, and the number of hidden nodes, H:

$$\begin{aligned} \beta &= 0.7(H)^{\frac{1}{I}} \\ &= 0.7\sqrt[I]{H}. \end{aligned}$$

Their procedure is based on the hyperbolic tangent activation function (cf. Section 4.3.3.3)

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The Nguyen-Widrow initialization procedure starts off by initializing all weights to random values between -γ and γ (commonly between -0,5 and 0,5). Denote the weight vector from the input nodes by $\mathbf{v}_h(\text{old})$ and compute

$$\| \mathbf{v}_h(\text{old}) \| = \sqrt{v_{h1}(\text{old})^2 + v_{h2}(\text{old})^2 + \ldots + v_{hI}(\text{old})^2}$$

For each hidden node the weights v_{hi} are then reinitialized as

$$v_{hi} = \frac{\beta v_{hi}(\text{old})}{\| \mathbf{v}_h(\text{old}) \|}.$$

The bias v_{h0} is set to a random number between $-\beta$ and β.

4.5.5 Batch training

One variation of the backpropagation algorithm to speed up learning is known as batch updating. Instead of updating weights after each training pattern is presented, weight updates are accumulated for several patterns, or even over an entire epoch, before being applied. A single weight adjustment, equal to the average of the accumulated weight correction terms, is then made for each weight. This may however, have a smoothing effect on the correction terms as the changes to the weights are correct on average, resulting in weights being skewed to the most recent patterns in the cycle ([Smith, 1993], [Zurada, 1992]). One solution is to choose patterns in a random sequence from a training set.

4.5.6 Steepness of the activation function

A neuron's continuous activation function is characterized by its slope parameter (cf. Section 4.3.3). Furthermore, as the derivative of the activation function is incorporated in the error information terms δ_j and δ_h, weights

are accordingly adjusted in proportion to the value of the derivative of the activation function. Both these factors - choice of activation function and choice of slope parameter - therefore strongly affects the learning process during backpropagation. Figure 4-11 displays the derivative of the bipolar sigmoid activation function (cf. Section 4.3.3.2) for four different values of the steepness parameter σ (0,5; 1; 2 and 4). This slope function illustrates how the steepness parameter σ affects the learning process.

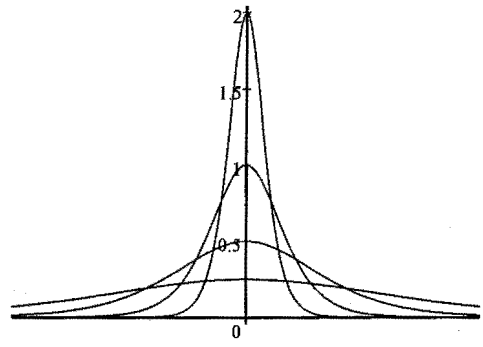

Figure 4-11: Bipolar sigmoid activation function slope for different σ

Firstly it can be observed that weights that are connected to nodes responding in their midrange are changed more than nodes that are already largely turned on or off. The local error information terms δ_j and δ_h are computed by multiplying with the derivative of the activation function, therefore the distributed components of the backpropagation error are large only for neurons in steep thresholding mode. Secondly, if the learning rate α is fixed, all weight adjustments are in proportion to the steepness coefficient σ. This implies that using activation functions with large slope parameters σ may give similar results to using large learning rates α. Rather than controlling both parameters, it is advisable to fix σ at a standard value of one (for the bipolar sigmoid activation function; for the binary sigmoid function the value will be

two), and to control learning speed solely by the learning rate α [Zurada, 1992].

4.5.7 The learning parameter α

The value of the learning parameter α is set by the user. The effectiveness and convergence of the error backpropagation algorithm depend to a large extent on the value of α. In general, however, α depends on the problem being solved thus different values of α work best on different training sets. Even though gradient descent is an efficient method for determining those weight values that minimize the squared error, error surfaces may possess properties that make the algorithm slow to converge. For example, if the error surface has broad minima, i.e. small gradient values, a larger value of α will result in a more rapid convergence of the procedure. Large values of α increases learning speed drastically. For problems with steep and narrow minima, however, a too large α will overshoot the solution, hence a small value of α must be selected. The learning rate should therefore be experimentally determined for each problem ([Zurada, 1992], [Smith, 1993]). It is important to remember that only small learning rates guarantee a true gradient descent. This is unfortunately offset by the increase in the number of epochs required to reach a satisfactory solution.

4.5.8 Backpropagation with momentum

Momentum is a common modification applied to standard backpropagation, i.e. online or stochastic training, to accelerate the convergence of the learning algorithm. Weight adjustments at each step are based on a combination of the current weight adjustment (as done in standard backpropagation) and the weight change from the previous step, with the result that the net is proceeding not in the direction of the gradient, but in the direction of a combination of the current gradient and the previous direction of weight correction

([Zurada, 1992], [Smith, 1993], [Fausett, 1994]). Backpropagation with momentum allows the network to ignore short-term fluctuations in the gradient. The advantage of this modification is experienced mainly when some training data are very different from the majority of the data. By incorporating the previous step's weight adjustment, a major disruption of the direction of learning is avoided when an unusual pair of training patterns are presented as the learning rate will be small. For relatively similar training data, training will be maintained at a fairly rapid pace.

The likelihood that the net will find weights that are a local instead of a global minimum is reduced with training by momentum. Momentum can be used in combination with batch training.

In backpropagation with momentum it is necessary to save the weight updates from more than one previous training patterns. In the simplest form, the new weights for training with momentum at step $t+1$ is based not only on the weights at step t, but also at step $t-1$. An additional term representing the momentum as a linear combination of the weight adjustments at times t and $t-1$ is added to the weight update formulas presented in Section 4.5.1.

For a momentum parameter μ, constrained to the range from 0 to 1, the weight update formula for the weights between the output and the hidden layer at time $t+1$ becomes

$$w_{jh}(t+1) = w_{jh}(t) + \alpha\delta_j z_h + \mu[w_{jh}(t) - w_{jh}(t-1)],$$

or

$$\Delta w_{jh}(t+1) = \alpha\delta_j z_h + \mu\Delta w_{jh}(t).$$

For the weights between the hidden and the input layer the weight update

formula at time $t+1$ becomes

$$v_{hi}(t+1) = v_{hi}(t) + \alpha\delta_h x_i + \mu[v_{hi}(t) - v_{hi}(t-1)],$$

or

$$\Delta v_{hi}(t+1) = \alpha\delta_h x_i + \mu\Delta v_{hi}(t).$$

A momentum constant between 0,1 and 0,8 is recommended by Zurada [Zurada, 1992].

4.5.9 Adaptive learning rates

Standard backpropagation, with each weight change based on all the examples and without momentum, is the only training algorithm that has been mathematically proven to converge on the set of weights producing minimum error. Weights are modified in the direction of the most rapid decrease of the error surface for the current weights. This does unfortunately not necessarily move the weights directly toward the optimal weight vector. One way to improve the speed of training for backpropagation is by adjusting the learning rate during training. Various algorithms have been developed for specific problems, e.g. for classification problems with totally unbalanced categories [Fausett, 1994]. One of the generally applied algorithms for adaptive learning is the delta-bar-delta algorithm, described in various textbooks (e.g. [Jacobs, 1988], [Fausett, 1994], [Bishop, 1995], [Haykin, 1999]).

The delta-bar-delta algorithm allows each weight to have its own learning rate. Learning rates are allowed to vary with time as training proceeds. Appropriate changes in the learning rate for each weight are determined by the direction of successive weight changes. If the weight changes are in the same direction, either an increase or a decrease, for several time steps, the learning

rate for that weight is increased. This condition is determined by the sign of the partial derivative of the error with respect to that weight for several time steps. For alternating partial derivative signs, indicating that the direction of weight changes alternates, however, the learning rate is decreased. These two heuristics do not guarantee improved network performance, although in practice adaptive learning rates invariably does. Furthermore, the delta-bar-delta modification may not always converge, but Jacobs provides a comparison of standard backpropagation, backpropagation with momentum and delta-bar-delta which shows that when training with the delta-bar-delta modification converges, it does so much faster, reducing training time by order of magnitude [Jacobs, 1988].

The delta-bar-delta algorithm consists of a weight update rule and a learning rate update rule.

Changes to the weights are as for the standard backpropagation algorithm, with the modification that each weight may change by a different proportion of the partial derivative of the error with respect to that weight. The weight vector change is thus no longer in the direction of the negative gradient.

Let E represent the squared error for the pattern presented at time t, let $w_{jh}(t)$ denote an arbitrary weight at time t (i.e. epoch t) and let $a_{jh}(t)$ be the learning rate for that weight at time t. The weight changes according to the delta-bar-delta notation is then

$$\begin{aligned} w_{jh}(t+1) &= w_{jh}(t) - a_{jh}(t+1)\frac{\partial E}{\partial w_{jh}} \\ &= w_{jh}(t) + a_{jh}(t+1)\delta_j z_h. \end{aligned}$$

Define a "delta" for each output node as:

$$\begin{aligned}\Delta_{jh} &= \frac{\partial E}{\partial w_{jh}} \\ &= -\delta_j z_h;\end{aligned}$$

and for each hidden node as

$$\begin{aligned}\Delta_{hi} &= \frac{\partial E}{\partial v_{hi}} \\ &= -\delta_h x_i.\end{aligned}$$

A combination of information about the current and the past derivative is then used to form a so-called "delta-bar" for each output node:

$$\overline{\Delta}_{jh}(t) = (1-\beta)\Delta_{jh}(t) + \beta\overline{\Delta}_{jh}(t-1),$$

and for each hidden node:

$$\overline{\Delta}_{hi}(t) = (1-\beta)\Delta_{hi}(t) + \beta\overline{\Delta}_{hi}(t-1),$$

with β, $0 < \beta < 1$, specified by the user.

The "delta-bar" terms are essentially weighted averages of past and current derivatives, while β can be considered as a weight on the past, accordingly $1-\beta$ is the weight put on the current derivative.

For example,

$$\beta = 0 \quad \text{gives} \quad \overline{\Delta}_{jh}(t) = \Delta_{jh}(t)$$

and the weights are simply adapted according to standard backpropagation as no past derivatives are considered.

$$\beta = 0,5 \quad \text{gives} \quad \overline{\Delta}_{jh}(t) = 0,5\Delta_{jh}(t) + 0,5\overline{\Delta}_{jh}(t-1).$$

As $\overline{\Delta}_{jh}(t-1)$ was in turn determined by past values of Δ_{jh}, one can write

$$\overline{\Delta}_{jh}(t) = 0,5\Delta_{jh}(t) + 0,25\Delta_{jh}(t-1) + 0,25\overline{\Delta}_{jh}(t-2),$$

and in turn

$$\overline{\Delta}_{jh}(t) = 0,5\Delta_{jh}(t) + 0,25\Delta_{jh}(t-1) + 0,125\Delta_{jh}(t-2) + 0,125\overline{\Delta}_{jh}(t-3),$$

and so forth, back to epoch 1. As β approaches 1, the current "delta" term (partial derivative of the weight with respect to the error) counts less while the past values of "delta" count more as these terms have been averaged into the "delta-bar" term.

The new learning rate is based on the premise that the learning rate should be increased if the weight changes are in the same direction on successive steps. It is effected by increasing the learning rate by a constant amount κ if $\overline{\Delta}_{jh}(t-1)$ and $\Delta_{jh}(t)$ have the same sign. Similarly, the learning rate is decreased by a by a proportion γ of its current value if $\overline{\Delta}_{jh}(t-1)$ and $\Delta_{jh}(t)$ have opposite signs.

The new learning rate is given by

$$\alpha_{jh}(t+1) = \begin{cases} \alpha_{jh}(t) + \kappa & \text{if} \quad \overline{\Delta}_{jh}(t-1) \cdot \Delta_{jh}(t) > 0, \\ (1-\gamma)\alpha_{jh}(t) & \text{if} \quad \overline{\Delta}_{jh}(t-1) \cdot \Delta_{jh}(t) < 0, \\ \alpha_{jh}(t) & \text{if} \quad \text{otherwise.} \end{cases}$$

Even though the delta-bar-delta algorithm requires the specification of the values of the three parameters β, $(0 < \beta < 1)$, κ and γ $(0 < \gamma < 1)$, the network results are in practice not sensitive to the choice. The values of $\kappa = 0,1$; $\gamma = 0,5$ and $\beta = 0,7$ are recommended by Smith as they work

well across a variety of problems [Smith, 1993]. Delta-bar-delta training can be combined with backpropagation using momentum, and provide benefits similar to momentum, noticeably much faster training.

4.5.10 Quickprop

Another fairly popular modification of backpropagation that accelerates the learning process is Quickprop, described in [Smith, 1993], [Fausett, 1994] and [Bishop, 1995], which was developed by Scott Fahlman [Fahlman, 1988]. The essential concept behind Quickprop is to include information about the curvature of the error surface as well as its slope to decide on weight changes. The modification is based on the assumptions that the error surface, as a function of each of the weights, can be approximated by a parabola that is concave and that the change in the gradient of the error curve for that particular weight is unaffected by other weights that are also changing. Quickprop uses information about the previous weight change and the value of gradient to determine the new weight change.

The value of the gradient at the t-th epoch is the sum of the partial derivatives of the error with respect to the given weight, summed over all P training patterns in the epoch:

$$S(t) = \sum_{p=1}^{P} \frac{\partial E(p)}{\partial w}.$$

Using the notation as for standard backpropagation, the slope from a hidden node to an output node is

$$S_{jh}(t) = -\sum_{p=1}^{P} \delta_j(p) z_h(p),$$

and the slope for the weight from an input node to a hidden node is

$$S_{hi}(t) = -\sum_{p=1}^{P} \delta_h(p) x_i(p).$$

The new weight change is defined as

$$\Delta w(t) = \frac{S(t)}{S(t-1) - S(t)} \Delta w(t-1).$$

The initial weight change can be taken as

$$\Delta w(0) = -\alpha S(0)$$

where α is the learning rate.

The behaviour of the algorithm must be considered for the three following cases [Fausett, 1994]:

1. If the current slope is in the same direction as the previous slope, but smaller, then the weight change will be in the same direction as in the previous step.

2. If the current slope is in the opposite direction from the previous slope, the weight change will also be in the opposite direction as in the previous step.

3. If the current slope is in the same direction as the previous slope, but is the same size or larger than the previous slope, the weight change will be infinite or the weights would be moved away from the minimum towards a maximum of the error.

Weight changes are therefore limited to prevent the difficulties associated with the third step.

To get the algorithm to work in practice often needs several restarts.

4.5.11 Training duration

The aim is, as is the case with most artificial neural networks, to train the net so that a balance is achieved between the ability to respond correctly to the input patterns that are used for training, i.e. memorization, and the ability to give reasonable responses to input that is similar, but not identical, to that used in training, i.e. generalization. It is therefore not necessarily advantageous to continue training the net until the total squared error actually reaches a minimum as overtraining may occur. A discussion on how to avoid overtraining by using a validation set has been discussed in Section 4.1.4.2.

4.5.12 Number of training pairs

A network with enough training patterns will be able to generalize to satisfaction (i.e. classify unknown testing patterns correctly). As discussed in Section 4.1.1, the VC-dimension which relates training set size, architecture and generalization performance helps to quantify the difficulty when learning from examples. One rule of thumb to determine the minimum number of training pairs P, based on e, the expected percentage correct classifications obtained by the trained net, is described in Fausett [Fausett, 1994]. The rule takes the relationship between the number of weights to be trained, say W, and the number of training patterns P into consideration. Specifically, to be able to classify $1 - e$ testing patterns correctly, the net must be trained to classify $1 - \frac{e}{2}$ of the training patterns correctly. The number of training patterns are then determined by the condition

$$\frac{W}{P} = e$$

or

$$P = \frac{W}{e}.$$

If the aim is to be able to classify 90% of the testing patterns correctly, $e = 0, 1$. An artificial neural network with 50 weights (between the input and hidden, and the hidden and output layers) will then require 500 training patterns assuming the net will be trained to classify 95% of the training patterns correctly.

It should be noted that these heuristics are guidelines only that will sometimes provide very conservative upper bounds on the number of samples or training pairs.

4.5.13 Implementation

The time spent on calculating weights using backpropagation is much longer than the time required to run the finalized artificial neural network in the recognition mode, which only involves the computations of the feedforward phase (Section 4.5.1 above). The network's weights are set to the final weights obtained from the backpropagation algorithm.

4.6 Techniques for evaluating the performance of a neural model

4.6.1 Validation

The weights of the neural model are adapted according to the values of the training data during the training phase. The objective of the training process is to minimize the mean squared error. If training continues to the point where

the squared error approaches zero, the model will have learned almost perfectly the characteristics of the training data, hence it overtrained.

R. Hecht-Nielsen suggested the use of an additional set of data, known as the validation set, during training to monitor the network's performance [Hecht-Nielsen, 1990]. The training and the validation sets are disjoint. At intervals during training, the error between the network output and the target output is computed using the validation set (cf. Section 4.1.4.2). If the error for the validation patterns decreases, training continues. However, as soon as the error begins to increase for the validation set, training is terminated as the net is at that stage starting to memorize the training patterns too specifically. Bearing in mind that the error may fluctuate, it is recommended that one tests for an increase in validation error over a window of epochs. A graph is often used to display the error calculated for the validation set.

4.6.2 Multiple random starts

The purpose of the training algorithm of a neural model is to move the system into a lower error state. The initial conditions of the model determine the starting point on the error surface. An unfortunate choice of initial conditions may result in a suboptimal solution. By changing these initial conditions repeatedly, a coarse exploration of the error surface is undertaken. It is recommended that for each neural model at least five experiments are undertaken). Typical validation results should be close together. Exploring the error surface in this way will aid in determining an optimal solution.

4.6.3 Recutting

If a large enough data set is available, multiple test files can be created. Each test file is trained and the performance of the neural model measured as the average over each test file. In the case of limited data, a number of different

training and test file cuts can be created from the original data. The overall model performance can be validated by measuring the average over each of the test files.

The advantage of recutting training and test files is that the best statistical estimate of the likely performance of the neural model is found by this averaging over a number of runs.

Chapter 5

Statistically optimizing the number of hidden nodes of an artificial neural network

Artificial neural network architecture selection (cf. Section 4.2) involves determining both the number of hidden layers in the artificial neural network and the appropriate number of hidden layer nodes. As discussed previously (cf. Section 4.2.2), a single hidden layer is sufficient to obtain an arbitrarily accurate approximation to an arbitrary mapping, provided that an adequate number of hidden nodes is available ([Cybenko, 1988], [Hornik & White, 1989], [Stinchcombe & White, 1989]). The number of hidden nodes is therefore a crucial parameter of a feedforward artificial neural network.

Generalization, the artificial neural network's ability to produce reasonable responses to patterns which are similar, but not identical, to the training data, is normally measured by the validation error. This is achieved by plotting the validation and training errors against the number of epochs, as explained in Chapter 4.

An indication that the system is beginning to learn the noise characteristics of the training data is seen when the validation error gradient, which initially drops at approximately the same rate as the training error during the early stages of learning, decreases more rapidly and then starts to increase [Hecht-Nielsen, 1990]. Overtraining, discussed in Section 4.1.1.1, is one of the serious problems encountered when using artificial neural networks for modelling. Possible solutions to the problem of overfitting is to limit the network's power by limiting the number of nodes, by limiting the number of epochs of training, i.e. stop training early, or by discouraging the network from using large weights ([Smith, 1993] , [Vapnik, 1995] and [Anthony & Bartlett, 1999]: Vapnik was in fact the first to note that the size of the parameter vector and the final layer weights are key factors to the generalization performance of a network).

On the other hand, if the validation error never goes up it may be an indication that the network does not have enough hidden nodes to overfit. This is also cause for concern, since it could mean that the network does not have enough hidden nodes to attain the necessary level of complexity to obtain an accurate model and hence may be underfitting the data.

Both these phenomena, overfitting and underfitting, emphasize the need to determine the optimal number of hidden nodes of an artificial neural network model as an important part of the modelling process.

Contrary to classical statistical procedures, the number of parameters in an artificial neural network is invariably comparable to the number of training patterns, necessitated by the need for a sufficiently complex model to adequately fit the data. Many different approaches have been investigated by artificial neural network users to rationally select the appropriate number of hidden layers and the optimum number of nodes in order to achieve the minimum generalization error. Amongst the more popular approaches to model se-

lection are Bayesian calculations of posteriors, regularization, cross-validation and the implementation of Kolmogorov complexity principles as well as code length measures of complexity [Vapnik, 1995].

Many of these methods essentially adapt the objective function of the training algorithm (cf. Section 4.4) by adding a term to it that penalizes architecture complexity. Complexity is automatically balanced against a close fit to the data when training takes place with an appropriate penalty term added to a term measuring the degree to which the network approximates the data. The objective is to minimize an expression which can be construed as a cost function that measures the degree of approximation of the model to the data that is added to a term measuring the complexity of the model. This complexity approach is based on an attempt to formalize the medieval maxim known as Occam's Razor which advocates adherence to the principle of sticking to the simplest, well-founded explanation, elucidated by E. Moody as "What can be done with fewer [assumptions] is done in vain with more." (quoted in [Vapnik, 1995]).

In this chapter a recursive algorithm - presented at the IEEE World Congress on Computational Intelligence in Alaska in 1998 [Fletcher & Engelbrecht, 1998] - is developed to statistically determine the optimal number of hidden nodes for an artificial neural network model by placing a statistical constraint on the reasonable complexity of the neural model. This is done by framing a feed-forward artificial neural network model with a single output node, trained by backpropagation, in a nonlinear regression setting. The mean squared error between the estimated network and a target function, which has a minimum with respect to the number of nodes in the hidden layer, is used to measure the accuracy of the artificial neural network model. By minimizing the mean square error, the algorithm combines artificial neural network training sessions with statistical analyses, using partitioned likelihood ratios, and an experimen-

tal design phase to generate new sessions until the optimal number of hidden nodes is reached. During the training process the excess number of hidden nodes at that stage, as well as the specific nodes to be pruned, are identified. This algorithm requires fewer sessions to establish the optimal number of hidden nodes than by using the straightforward way of eliminating nodes successively one by one, as is often the case in practice.

5.1 Problem setting

5.1.1 A feedforward artificial neural network in a nonlinear regression setting

Define a univariate response nonlinear model with additive noise as

$$\mathbf{t}(\mathbf{x}_p) = g(\mathbf{x}_p) + \mathbf{e}(\mathbf{x}_p) \qquad \text{with } p = 1, 2, ..., P \tag{5.1}$$

where the data $\mathbf{x}_p \epsilon \Re^I$ and $\mathbf{e}(\mathbf{x}_p) \overset{IID}{\sim} N(0, \sigma^2)$ [Green & Silverman, 1994].

As mentioned above, any continuous mapping can be accurately approximated by a single hidden layer. The unknown nonlinear function $g(\mathbf{x}_p)$ can therefore be fitted to the data $(\mathbf{x}_p, \mathbf{t}(\mathbf{x}_p))$ over all P training patterns, or examples, by a one hidden layer, feedforward artificial neural network with H hidden nodes. In matrix notation the model is expressed as

$$\mathbf{o}(\mathbf{x}_p, \mathbf{v}_H, \mathbf{w}) = \varphi(\mathbf{z}(\mathbf{x}_p, \mathbf{v}_H, \mathbf{w})) \tag{5.2}$$

with individual elements

$$o_p = \varphi\left(\sum_{h=1}^{H} w_h z_{hp} + w_0\right)$$

where

$$z_{hp} = \phi(\sum_{i=1}^{I} v_{hi} x_{ip} + v_{h0})$$

and I is the total number of input nodes (independent variables or predictors in statistical terminology). The total number of parameters (weights) to be estimated is $H(I+1)+(H+1)$.

Here $\mathbf{o} = (o_p)$:$P \times 1$ is the output vector of the artificial neural network; $\mathbf{X} = (x_{ip}) : I \times P$ is the p-dimensional matrix of input variables; $\mathbf{V} = \mathbf{v}_H = (v_{hi})$ is the $H \times (I+1)$ weight matrix where v_{hi} is the weight connecting node h of the hidden layer to the i-th input variable and v_{h0} is the weight between the h-th hidden node and the bias unit; $\mathbf{w}$ is the $(H+1)$-dimensional weight vector between the hidden layer and the output layer, including the bias weight w_0; ϕ is a sigmoid function and φ is either a linear or a sigmoid function.

Diagrammatically, this neural model can be displayed as in Figure 5-1.

In terms of approximation theory and Statistics, fitting the data using model (5.2) is a parametric nonlinear regression problem specified by the structure of the model and the sigmoid function used in it ([Barron, 1993], [Barron, 1994], [Cheng & Titterington, 1994], [Vapnik, 1995]). For the problem with additive noise, as defined in (5.1), the optimal number of parameters (i.e. hidden nodes in the artificial neural network model) can be found, based on an accuracy analysis.

5.1.2 Accuracy criteria

The model defined in (5.2) is used to reconstruct the unknown nonlinear function $g(\mathbf{x}_p)$ from the observations $\mathbf{x}_p$. The quality of the reconstruction, i.e. the accuracy of the model, is characterized by the mean squared error (MSE) risk

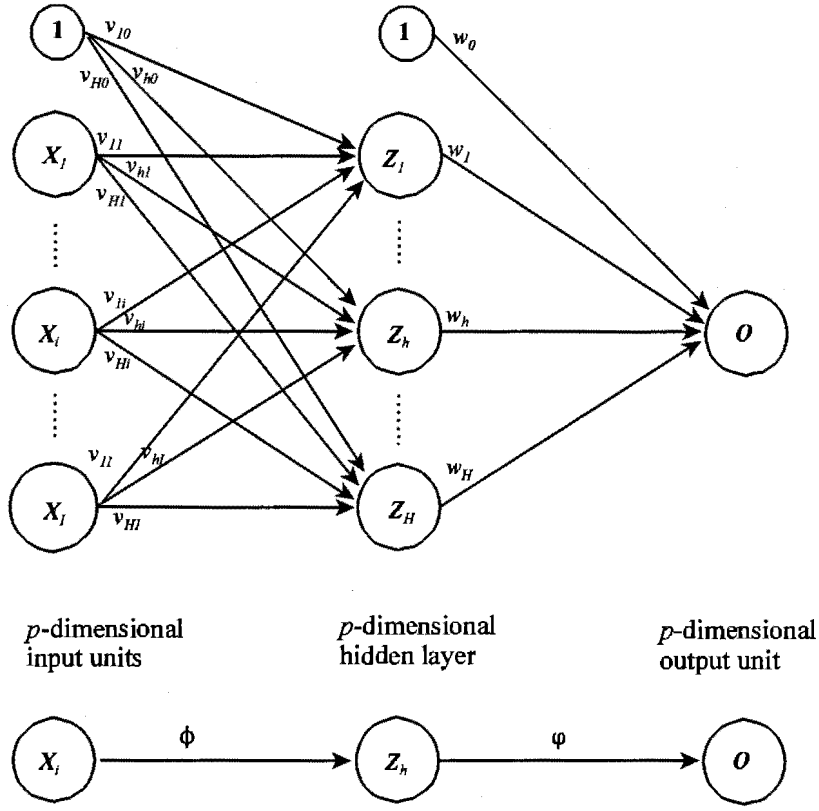

Figure 5-1: A feedforward artificial neural network with a single output node

function. This function, an accuracy criterion, is defined by

$$\mathbf{R}_{P,H}(\mathbf{v}_H, \mathbf{w}) = \frac{1}{P} \sum_{\mathbf{x}_p} \mathbf{E}(g(\mathbf{x}_p) - \mathbf{o}(\mathbf{x}_p, \widehat{\mathbf{v}}_H, \widehat{\mathbf{w}}))^2 \tag{5.3}$$

where $\widehat{\mathbf{v}}_H$ and $\widehat{\mathbf{w}}$ are estimates of $\mathbf{v}_H$ and $\mathbf{w}$ respectively, obtained from the artificial neural network model. $\mathbf{E}$ denotes the expectation.

The set of optimal parameters $\mathbf{v}_H^*$ and $\mathbf{w}^*$ which will minimize $\mathbf{R}_{P,H}$ with respect to $\mathbf{v}_H$ and $\mathbf{w}$ is

$$(\mathbf{v}_H^*, \mathbf{w}^*) = \arg \min_{\mathbf{v}_H, \mathbf{w}} (\mathbf{R}_{P,H}).$$

This risk or error function is also known as the cost function or objective function.

It has been proved that the artificial neural network model given by (5.2) provides a very good uniform approximation to quite a wide class of functions $g(\mathbf{x})$ [Barron, 1994]. This means that under some general conditions

$$\mathbf{R}_{P,H}(\mathbf{v}_H^*, \mathbf{w}^*)) \to 0 \text{ as } H \to \infty.$$

Obviously an excessive increase in the number of nodes H in the hidden layer is undesirable because it adds excessively to the complexity of the model, with the accompanying increasing costs involved in the training of the artificial neural network.

There is, however, another restriction on the level H of the reasonable complexity of the neural model, determined by the stochastic nature of the noisy data of the specific problem (as stated in (5.1)) under consideration. Generally speaking, for any given number of observations, an increasing number of nodes H will result in an increase in the variance of the estimates of the (weight) parameters $\mathbf{V}$ and $\mathbf{w}$ of the model, and consequently also of the estimation errors and the accuracy criterion. As a result, including more nodes H after some level H^* becomes unjustified because of the random uncertainty inherent in the data.

Barron has shown that the two contributions to the total MSE risk (5.3) are the approximation error and the estimation error [Barron, 1993], [Barron, 1994]. The approximation error of fitting the artificial neural network model refers to the distance between the target function and the closest artificial neural network function of a given architecture (i.e. the bias), while the estimation error reflects the variance of the random error, i.e. the difference between this ideal, closest artificial neural network function and an estimated network function.

Asymptotically, as $P \to \infty$, subject to some unrestrictive assumptions

about $g(\mathbf{x}_p)$,

$$\mathbf{R}^*_{P,H} = \mathbf{R}_{P,H}(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}}) \approx \frac{c_1}{H^*} + c_2 \frac{H^*}{P} \log(P) \tag{5.4}$$

where c_1 and c_2 are constants [Barron, 1994].

The first term on the right-hand side of (5.4) corresponds to the approximation error, or bias, of fitting the artificial neural network and the second one to the variance of the random errors. Hence the optimal number of nodes H^* represents a bias-variance trade-off. Overfitting, i.e. $H > H^*$, thus not only involves extra computation but, as pointed out above, is also undesirable from the point of view of the accuracy achieved: generalization performance is degraded in terms of artificial neural network theory. Note that as the number of observations (training patterns) P increases, the optimum number of nodes H^* also increases proportionally.

Minimizing function (5.4) with respect to H^* will give the optimal number of nodes. The derivative of function (5.4) with respect to H^* is

$$\frac{\partial \mathbf{R}^*_{P,H}}{\partial H^*} = -\frac{c_1}{(H^*)^2} + c_2 \frac{\log P}{P}.$$

Setting this equation to zero gives

$$\frac{c_1}{(H^*)^2} = c_2 \frac{\log P}{P}$$

$$\text{i.e.} \qquad (H^*)^2 = \frac{c_1}{c_2} \cdot \frac{P}{\log P}.$$

Hence the optimal number of nodes is obtained by

$$H^* = \sqrt{\left(\frac{c_1}{c_2} \cdot \frac{P}{\log P}\right)}. \tag{5.5}$$

The challenge is to optimize an artificial neural network based on the accuracy criteria set out in (5.3) and (5.4).

One of the best-known and most widely used sampling accuracy criteria is the mean squared error.

Let $(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})$ be the set of weights found by the artificial neural network found as a solution in a standard training session to the least squares problem

$$\mathbf{Q}_{P,H}(\mathbf{v}_H, \mathbf{w}) = \frac{1}{P} \sum_{\mathbf{x}_p} (\mathbf{t}(\mathbf{x}_p) - \mathbf{o}(\mathbf{x}_p, \mathbf{v}_H, \mathbf{w}))^2 \tag{5.6}$$

i.e. $\mathbf{Q}_{P,H}$ is an empirical risk and

$$(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}}) = \arg \min_{\mathbf{v}_H, \mathbf{w}} \mathbf{Q}_{P,H}(\mathbf{v}_H, \mathbf{w}). \tag{5.7}$$

The empirical risk $\mathbf{Q}_{P,H}(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})$ is a monotonically decreasing function of H. It therefore does not have a minimum with respect to H and cannot simply be used instead of $\mathbf{R}^*_{P,H}$ (cf. (5.4)). This situation is quite typical of statistical problems dealing with model selection and in particular with selection of the complexity of the model.

A great amount of effort has been expended to find good approximations for $\mathbf{R}_{P,H}(\mathbf{v}_H, \mathbf{w})$ and (5.6) - (5.5) in the form

$$\widehat{H} = \arg \min_H (\log \mathbf{Q}_{P,H}(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}}) + \psi(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})) \tag{5.8}$$

where $\psi(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})$ is a penalty function increasing with the complexity (i.e. number of nodes) of the model (e.g. [Hurrich & Tsai, 1989], [Wei, 1992], [Hassoun, 1995], [Fine, 1999]).

It has been shown that methods such as cross-validation, generalized cross-validation, Akaike and the C_p criteria differ only by the penalty function $\psi(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})$ in (5.8) (e.g. [Hurrich & Tsai, 1989], [Wei, 1992], [White, 1989],

[White, 1992], [Cohen, 1995]). Procedure (5.8) is computationally efficient, but the function $\psi(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})$ and the approach on the whole are well justified only for linear regression models.

Loss functions having features of (5.8) have been developed specifically for artificial neural networks ([Barron, 1993], [Barron, 1994], [Vapnik, 1995]). The proposed penalty functions include unspecified parameters and functions which enable (5.8) to have a minimum on H. However the practical applications of these methods are questionable due to the above-mentioned ambiguity embedded in $\psi(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})$.

Cross-validation methods (e.g. [Amari *et al.*, 1995], [Hassoun, 1995]) in their original combinatorial form are in many cases able to provide good approximations to problem (5.4) but are computationally intensive.

The "one-out" cross-validation loss function has the form

$$\mathbf{Q}_{P,H}^{[CV]} = \frac{1}{P_C} \sum_{\mathbf{x}_p \epsilon \mathbf{X}_C} (\mathbf{t}(\mathbf{x}_p) - \widehat{\mathbf{o}}(\mathbf{x}^{[p]}, \mathbf{v}_H^{[p]}, \mathbf{w}^{[p]}))^2 \tag{5.9}$$

where

$$(\mathbf{v}_H^{[p]}, \mathbf{w}^{[p]}) = \arg \min_{\mathbf{v}_H, \mathbf{w}} \frac{1}{P-1} \sum_{r \neq p} (\mathbf{t}(\mathbf{x}_r) - \widehat{\mathbf{o}}(\mathbf{x}_r, \mathbf{v}_H, \mathbf{w}))^2, \tag{5.10}$$

$\mathbf{X}_C$ is a control set, P_C is the number of observations in it, and a permutation over a set of observations is assumed in (5.9) - (5.10).

A simplified form of cross-validation where the observations are split into two sets - one for learning and one for control – is usually quite practical and good results are obtained when the number of observations is much larger than the number of parameters to be estimated.

A different approach, based on a statistical analysis of the results of training sessions, is proposed in this chapter.

Consider the empirical risk $\mathbf{Q}_{P,H}(\widehat{\mathbf{v}}_H, \widehat{\mathbf{w}})$ in (5.6) with parameters $\widehat{\mathbf{v}}_H$ and $\widehat{\mathbf{w}}$ found from (5.7) as a function of H. When the model is substantially overfitted the bias components are compensated for, however the sum of random errors becomes large.

A statistical test can be used to evaluate $\mathbf{Q}_{P,H}(\mathbf{v}_H, \mathbf{w})$ with decreasing H to find the critical region of $\widehat{H}$ where the bias becomes non-negligible. The critical value of $\widehat{H}$ does not actually produce the compromise bias-variance in (5.4) but simply serves to identify an area of H values where further decreasing the number of nodes becomes counter-effective, while an increase does not improve the generalization abilities of the artificial neural network.

Depending on the significance level α used in the statistical tests, the artificial neural network is either overfitted ($\widehat{H} > H^*$) or underfitted ($\widehat{H} < H^*$). This statistical test approach can consequently be treated as an approximate solution to (5.4) - (5.5).

5.2 Statistical theory and method

The statistical concepts and theory that are relevant to the algorithm which will be used to determine the optimal number of hidden nodes of a feedforward artificial neural network in a nonlinear regression setting as defined in (5.1) - (5.2) are briefly introduced in this section and explained in the artificial neural network setup.

Basically, the algorithm specifies the conditional statistical analysis of a completed training session of the artificial neural network to determine a statistically justified number of nodes for the following training session, as well as specifying the particular nodes that can be pruned. The term "conditional" means here that in the statistical analysis the weights $\mathbf{v}_H$ between the input units and the hidden node layer are assumed to be fixed ($\mathbf{v}_H = \widehat{\mathbf{v}}_H$), while the

weights $\mathbf{w}$ between the hidden layer and the output node are allowed to vary.

5.2.1 Statistical decision-making

The statistical decision-making process comprises a logical number of steps: the design of the experiment, the execution of the experiment and finally the analysis of the results obtained by the experiment. Depending on the setting of the problem, this process may be iterative. If warranted, the experiment may be redesigned, re-executed and re-analysed.

In an artificial neural network setting, the process of determining the architecture of the network can be viewed as the experimental design stage. For a feedforward artificial neural network with one hidden layer, this entails determining the number of nodes in the hidden layer.

An artificial neural network training session that results in obtaining estimates of the weight parameters $\mathbf{v}_H$ and $\mathbf{w}$ can be considered as the experiment. The specific feature of the experiment is using a complex learning algorithm (in this case backpropagation) to provide estimates of the weight parameters. An experiment need not necessarily be a single training session, but can be complex, and in particular, can include a number of attempts with different initial conditions for the backpropagation algorithm.

During the analysis stage the results of the experiment are evaluated, possibly for use in the design of the next experiment.

An initialization process is needed to start the process in the case of artificial neural networks.

5.2.2 Fisher information matrix

The univariate response nonlinear model in (5.1) is usually expressed in statistical notation as

$$\mathbf{y} = g(\mathbf{Z}, \boldsymbol{\beta}) + \boldsymbol{\epsilon},$$

with individual elements

$$y_p = g(\mathbf{z}_p, \boldsymbol{\beta}) + \epsilon_p.$$

$\mathbf{Z}$ is the matrix of p-dimensional regressors or independent variables, $\boldsymbol{\beta}$ is the H-dimensional vector of parameters to be estimated, and $\boldsymbol{\epsilon} \overset{IID}{\sim} N(0, \sigma^2)$.

(The notation $\mathbf{X}$, instead of $\mathbf{Z}$, is more common in statistical textbooks, but is not adopted here to avoid confusion between the artificial neural network input matrix $\mathbf{X}$ defined in Section 5.1.1 and the constrained artificial neural network input $\mathbf{Z}$ to the standard linear regression model explained in the next section.)

The inverse of the Fisher information matrix is used to obtain an estimate of the asymptotic variance-covariance matrix of the parameter estimates $\widehat{\boldsymbol{\beta}}$ and is used to assess the quality of the estimators. The estimated parameters of the information matrix is obtained at convergence and are as follows [Ratkowsky, 1983]:

$$\Phi = \frac{1}{\widehat{\sigma}^2} \sum_{h=1}^{H} \frac{\partial g(\mathbf{z}_p, \widehat{\boldsymbol{\beta}})}{\partial \widehat{\beta}_i} \cdot \frac{\partial g(\mathbf{z}_p, \widehat{\boldsymbol{\beta}})}{\partial \widehat{\beta}_j}. \tag{5.11}$$

Each diagonal element of the inverse of the information matrix is a lower bound for the variance for the corresponding parameter ([Ratkowsky, 1983],

[Deco & Obradovic, 1996]), i.e.

$$cov(\widehat{\boldsymbol{\beta}}) \cong \Phi^{-1}\widehat{\sigma}^2.$$

Under the above-mentioned conditions for $\boldsymbol{\epsilon}$, $\widehat{\sigma}^2$ is simply the mean squared errors (analogous to (5.6)).

For the constrained artificial neural network model, i.e. where the weights $\mathbf{v}_H = \widehat{\mathbf{v}}_H$ are presumed fixed as explained above, the elements of the Fisher information matrix 5.11 correspond directly to the partial derivatives of the artificial neural network output with respect to the individual weights w_h connecting the hidden layer nodes $\mathbf{z}_h$ to the output node $\mathbf{o}$. These partial derivatives are calculated during the backpropagation stage of training (cf. Section 4.5.1).

In this case the conditional Fisher information matrix $(H \times H)$ is defined as

$$\Phi(\widehat{\mathbf{v}}_H, \mathbf{w}) = \frac{1}{P}\sum_{\mathbf{x}_p} \frac{\partial o_p(\mathbf{x}_p, \widehat{\mathbf{v}}_H, \mathbf{w})}{\partial \mathbf{w}} \cdot \frac{\partial o_p(\mathbf{x}_p, \widehat{\mathbf{v}}_H, \mathbf{w})}{\partial \mathbf{w}^T}. \tag{5.12}$$

(The artificial neural network model actually has $H + 1$ weights for the H hidden nodes plus the bias in the hidden layer which is a general constant that specifies the intercept and aids in the positioning of the model. Interest is restricted here to the H weights or parameters that relate to fitting the nonlinear function describing the model. The bias unit will not be pruned by the algorithm.)

To obtain the estimated variance-covariance matrix of the estimated weight parameters $\widehat{\mathbf{w}}$, this matrix of partial derivatives, which is square and symmet-

ric, must be inverted:

$$cov(\widehat{\mathbf{w}}) = (\Phi(\widehat{\mathbf{v}}_H, \mathbf{w}))^{-1} \widehat{\sigma}^2 \tag{5.13}$$

$$= \mathbf{C}^{-1}\widehat{\sigma}^2$$

where $\mathbf{C}$ is the conditional Fisher information matrix.

5.2.3 Standard linear regression model

Artificial neural network models in a nonlinear regression setting uses a sigmoid function only in the hidden layer to calculate z_{hp} from the input units x_{ip} and the weights v_{ji} connecting these two layers (cf. (5.2) and Figure 5-1). Thus by constraining the model by considering only the output weights w_h, the artificial neural network model (5.2) can also be phrased exactly in the framework of the standard linear regression model:

$$\mathbf{y} = \mathbf{Z}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{5.14}$$

where the vector $\mathbf{y}$ corresponds to the output vector $\mathbf{o}(\mathbf{x}_p, \mathbf{v}_H, \mathbf{w})$ of the artificial neural network model, the parameter vector $\boldsymbol{\beta}$ corresponds to the artificial neural network model weight vector $\mathbf{w}$ connecting the hidden layer with the single output node $\mathbf{o}$ and matrix $\mathbf{Z} = (z_{hp}) : P \times (H+1)$ is defined as the output of the hidden nodes:

$$\mathbf{Z} = \begin{bmatrix} 1 & \phi(\sum_{i=0}^{I} \widehat{v}_{i1} x_{1i}) & \phi(\sum_{i=0}^{I} \widehat{v}_{i2} x_{1i}) & \dots & \phi(\sum_{i=0}^{I} \widehat{v}_{iH} x_{1i}) \\ 1 & \phi(\sum_{i=0}^{I} \widehat{v}_{i1} x_{2i}) & \phi(\sum_{i=0}^{I} \widehat{v}_{i2} x_{2i}) & \dots & \phi(\sum_{i=0}^{I} \widehat{v}_{iH} x_{2i}) \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \phi(\sum_{i=0}^{I} \widehat{v}_{i1} x_{Pi}) & \phi(\sum_{i=0}^{I} \widehat{v}_{i2} x_{Pi}) & \dots & \phi(\sum_{i=0}^{I} \widehat{v}_{iH} x_{Pi}) \end{bmatrix} \tag{5.15}$$

with ϕ, $\widehat{v}_{ij}$ and x_{ji} defined in Section 5.1.1.

As in the previous section, interest is restricted only to the H parameters that relate to fitting the model. The $H \times H$ variance-covariance matrix of the least squares parameter estimators $\widehat{\boldsymbol{\beta}}$ (i.e. of the estimated weight parameters $\widehat{\mathbf{w}}$) of the linear regression model in (5.14) is [Bates & Watts, 1988]

$$cov(\widehat{\boldsymbol{\beta}}) = (\mathbf{Z}^T\mathbf{Z})^{-1}\widehat{\sigma}^2$$

i.e.

$$\begin{aligned} cov(\widehat{\mathbf{w}}) &= (\mathbf{Z}^T\mathbf{Z})^{-1}\widehat{\sigma}^2 \qquad (5.16) \\ &= \mathbf{C}^{-1}\widehat{\sigma}^2. \end{aligned}$$

For the constrained artificial neural networks the two matrices (5.13) and (5.16) are therefore identical, i.e. the conditional Fisher information matrix $\Phi(\widehat{\mathbf{v}}_H, \mathbf{w})$ is equal to $\mathbf{Z}^T\mathbf{Z}$.

5.2.4 Singular value decomposition

Any rectangular $N \times M$ matrix $\mathbf{Q}$ of rank R can be uniquely decomposed (up to a simultaneous reflection of corresponding columns of $\mathbf{U}$ and $\mathbf{V}$) in the form $\mathbf{Q} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ where matrices $\mathbf{U} : N \times R$ and $\mathbf{V} : M \times R$ have orthonormal columns (i.e. $\mathbf{U}^T\mathbf{U} = \mathbf{V}^T V = \mathbf{I}_{(R)}$) and $\mathbf{D}$ is a diagonal matrix with nonnegative entries. This decomposition is known as the singular value decomposition of the matrix and is useful to determine the rank of the matrix. If $\mathbf{Q}$ is square symmetric, $\mathbf{U} = \mathbf{V}$ and the technique gives the eigenstructure or spectral decomposition of the matrix [Green & Carroll, 1978].

The singular value decomposition of the estimated variance-covariance ma-

trix $\mathbf{C} : H \times H$ of the parameter estimates $\widehat{\mathbf{w}}$ is

$$\mathbf{C} = \mathbf{U}\mathbf{D}_\lambda\mathbf{U}^T \tag{5.17}$$

$$\mathbf{D}_\lambda = \text{diag}(\lambda_1, \ldots, \lambda_{H-1}, \lambda_H) \geq 0$$

with the eigenvalues λ_h in descending order. The elements of $\mathbf{D}_\lambda$ can be interpreted as some measure of the components of the variance-covariance matrix and will be used to identify parameters (i.e. weights w_h corresponding to the hidden nodes $\mathbf{z}_h$) which make a small contribution to the neural model, as explained in the next Section.

5.3 Optimization algorithm

The recursive algorithm determines the critical area of $\widehat{H}$ which renders an approximate solution to the bias-variance trade-off in (5.4) where further decreasing the number of nodes becomes counter-effective, while an increase does not improve the generalization abilities of the artificial neural network. The artificial neural network is thus optimized in terms of the number of hidden nodes.

The algorithm comprises training sessions of the artificial neural network, statistical analyses of the results and the experimental design to determine the artificial neural network architecture in subsequent sessions. The aim is to minimize the number of training sessions necessary to reach an optimal the number of hidden nodes.

The approximation procedure adheres to the process of statistical decision-making (cf. Section 5.2.1).

5.3.1 Description of the algorithm

5.3.1.1 Initialization

The artificial neural network is trained with an initial architecture of H hidden nodes (plus a bias unit for the intercept) as determined by heuristic rules.

5.3.1.2 Experimental design

The main procedure of the design phase is a local conditional analysis during which the artificial neural network model for the experiment is specified. The purpose of this analysis is two-fold: to determine a statistically justified number of hidden nodes, $\triangle H$, which can conceivably be eliminated from the network, as well as specifying the particular nodes that can be pruned. By specifying these $\triangle H$ nodes, the weight values achieved for the remaining nodes from the previous session are preserved and can therefore be used in the next stage.

This implies a two-stage procedure after the network has been trained: the first stage for constructing a hypothesis about $\triangle H$ and the corresponding specified nodes, and the second stage to test this hypothesis.

5.3.1.2.1 Stage I The multiple hypotheses to be tested have the form

$$H_{0q} : w_k = 0 \quad \forall \quad k \in K_q \tag{5.18}$$

where K_q, $q = 1, .., Q$, are subsets of nodes considered for elimination from the artificial neural network.

The sets K_q in (5.18) are constructed by an analysis of the singular value decomposition (cf. Section 5.2.4) of the conditional Fisher information matrix (introduced in Section 5.2.2) and differ by the number of nodes considered for elimination. The number of nodes available for pruning is determined by the

nonzero output weights obtained from the artificial neural network training session.

The singular value decomposition (or eigenvalue decomposition) of the conditional Fisher information matrix $\Phi(\widehat{\mathbf{v}}_H, \mathbf{w})$ (5.12), is given by

$$\mathbf{Z}^T\mathbf{Z} = \mathbf{U}\mathbf{D}_\lambda\mathbf{U}^T, \tag{5.19}$$

$$\mathbf{D}_\lambda = \text{diag}(\lambda_1, ..., \lambda_H)$$

The group of smallest eigenvalues $\lambda_h, \lambda_{h+1}, ..., \lambda_H$ that differs in order from the other eigenvalues determines the number of nodes which makes a small contribution to the neural model. The number of eigenvalues in this set determines the value of $\triangle H = H - h$, thus accomplishing the first aim of Stage I. If $\triangle H = 0$ the model cannot be reduced.

Denote the N-th column of matrix $\mathbf{U}$ by $\mathbf{u}_N$. If the absolute values of the elements of $\mathbf{u}_N$ are very different, then the largest of these elements, say u_{N_i} and u_{N_j}, indicate the suspect nodes, say n_i and n_j, that can be excluded from the artificial neural network. This property is used in order to form the sets K_q in (5.18) corresponding to the groups of the smallest eigenvalues in $\mathbf{D}_\lambda$.

5.3.1.2.2 Stage II In order to test the hypothesis (5.18) that a suspect set of nodes are redundant for any given K_q, the standard likelihood-ratio test statistic L is used to determine if there is a statistically significant difference between the reduced model (with $H - \triangle H$ nodes in the hidden layer) and the full model (with H nodes in the hidden layer):

$$L = \left(\frac{SSE_R - SSE_F}{df_R - df_F}\right) \Big/ \left(\frac{SSE_F}{df_F}\right) \tag{5.20}$$

Under the null hypothesis that the constrained model with $\triangle H$ fewer hid-

den nodes is equivalent to the model with H hidden nodes, the test statistic $L \sim F(df_R - df_F; df_F)$. The degrees of freedom for the full model is

$$df_F = IP - H$$

and the degrees of freedom for the reduced model is

$$df_R = IP - (H - \triangle H_q)$$

where $\triangle H_q$ is the number of weights in the set K_q.

SSE_F and SSE_R are the sum of squared errors for the full model and the reduced model respectively.

The artificial neural network is not trained during this stage as the submatrix needed to calculate the differences between the target values t_p and the constrained artificial neural network output can simply be obtained from the full model trained during the current session.

The least squares estimator of $\mathbf{w}_F$ is the $H \times 1$ weight vector $\widehat{\mathbf{w}}_F$ that minimizes the sum of squared errors with respect to $\mathbf{w}$ and is given as

$$\widehat{\mathbf{w}}_F = \arg\min_{\mathbf{w}} SSE_F$$

where

$$SSE_F = \min_{\mathbf{w}} \sum_p (t_p - o(\mathbf{x}_p, \widehat{\mathbf{v}}_H, \mathbf{w}))^2. \tag{5.21}$$

The weight vector $\widehat{\mathbf{w}}_F$ is therefore simply the weight vector $\widehat{\mathbf{w}}$ obtained from training the artificial neural network using the backpropagation algorithm. For this constrained model, the matrix $\mathbf{Z}$ as defined in (5.15) is the output of the hidden layer of the artificial neural network, consequently the sum of squared

errors for the full model can be calculated as

$$SSE_F = \| \mathbf{t} - \mathbf{Z}\widehat{\mathbf{w}}_F \| .$$

The reduced weight vector and the sum of squared errors for the reduced model are obtained by excluding the suspected redundant weights from the calculations:

$$\widehat{\mathbf{w}}_R = \arg\min_{\mathbf{w}} SSE_R$$

and

$$\begin{aligned} SSE_R &= \min_{\mathbf{w}_{H-\Delta H_q}} \sum_p (t_p - o(\mathbf{x}_p, \widehat{\mathbf{v}}_H, \mathbf{w}_{H-\Delta H_q}))^2 \qquad (5.22) \\ &= \| \mathbf{t} - \mathbf{Z}_R \widehat{\mathbf{w}}_R \| \end{aligned}$$

using the same reasoning as for the full model. Here $\widehat{\mathbf{w}}_R : (H - \Delta H_q) \times 1$ is denoted by $\mathbf{w}_{H-\Delta H_q}$.

When L exceeds the α-critical point of the F-distribution with said degrees of freedom, H_{0q} in (5.18) is rejected, i.e. the reduced model contains hidden nodes that are not redundant. It is therefore necessary to identify and define a different set of ΔH suspect nodes to be tested for redundancy. These different sets of suspect nodes to be eliminated from the model can all be tested by statistic (5.20), with sums of squared errors defined by (5.21) and (5.22).

5.3.1.3 Experiment

Once the architecture for the reduced model has been established during the design phase, the artificial neural network is trained using the reduced number of hidden nodes to produce a set of parameter estimates for the weights $\mathbf{v}_H$

and $\mathbf{w}$ of the reduced model. These results are analysed during the next stage.

5.3.1.4 Analysis

The analysis stage involves the comparison of the experimental results where the two successive training sessions are compared, i.e. the output of the reduced model is compared to that of the full model. The hypothesis postulates that the reduced model with $H - \triangle H$ nodes is equivalent to the full model with H hidden nodes. If this hypothesis is rejected, the reduced model is rejected in favour of the full model. This means that too many nodes have been pruned from the hidden layer and it is necessary to return to the design phase to determine a different (smaller) $\triangle H$ set of suspect nodes to be investigated using the local conditional analysis as described in Section 5.3.1.2.

The hypothesis is tested using the likelihood-ratio test statistic L as defined in (5.20):

$$L = \left(\frac{SSE_R - SSE_F}{df_R - df_F}\right) \Big/ \left(\frac{SSE_F}{df_F}\right)$$

As above, the statistic L has an F-distributed under the null hypothesis with $df_R - df_F$ numerator and df_F denominator degrees of freedom.

When $L \leq F_{\alpha;df_R-df_F;df_F}$, the reduced model with $H - \triangle H$ nodes is used in the conditional analysis as described in the design phase (Section 5.3.1.2) to establish if more nodes can be pruned from the hidden layer.

Although the test statistic used at this step is identical in form to (5.20), it is completely different in nature as the sums of squared errors are calculated entirely differently. Here, SSE_F and SSE_R, the sum of squared errors for the full and the reduced models respectively, are calculated as the sum of squared differences between the target values t_p and the artificial neural network output

trained for the two different models obtained from the experiment:

$$SSE_F = \sum_{\mathbf{x}_p} (\mathbf{t}(\mathbf{x}_p) - \mathbf{o}(\mathbf{x}_p, \mathbf{v}_H, \mathbf{w}))^2 \tag{5.23}$$

and

$$SSE_R = \sum_{\mathbf{x}_p} (\mathbf{t}(\mathbf{x}_p) - \mathbf{o}(\mathbf{x}_p, \mathbf{v}_H, \mathbf{w}_{H-\Delta H}))^2 \tag{5.24}$$

The number of degrees of freedom for the full model is

$$df_F = (I+1)P - ((I+1) \times (H+1) + (H+1))$$

and for the reduced model it is

$$df_R = (I+1)P - ((I+1) \times (H-\triangle H+1) + (H-\triangle H+1)),$$

i.e. $df_R - df_F$ gives the number of weight parameters which are constrained to zero in the reduced model.

5.3.2 Related research

This algorithm is different in a number of aspects from previously reported results.

Research closely related to this work is architecture selection of an artificial neural network as reported by Steppe et al where the likelihood-ratio test statistic is used as a model selection criterion to compare artificial neural networks with a decreasing number of nodes [Steppe & Rogers, 1996]. The criterion is used in a sequential procedure where each successive model differs by one hidden node only. According to their algorithm, the artificial neural network, starting from an initial architecture with large H, requires a full

training procedure for each successive model until $\widehat{H}$ is found. In the above algorithm more than one node can be pruned at every step of the descent from H to $\widehat{H}$, hence successive sessions can differ by more than one node, resulting in a substantial saving in the amount of training sessions.

Xue et al. reported using the singular value decomposition in an attempt to reduce the dimensionality formed by the hidden nodes [Xue & Tompkins, 1990]. They determined the rank of the output variance-covariance matrix of the hidden nodes of a back-propagation model using the singular value decomposition to decide on the appropriate number of hidden nodes, and then calculated the correlation coefficients of the weight matrix consisting of the connections to the hidden layer in order to determine which nodes are redundant. Contrary to their method, the algorithm set out below does not base the singular value decomposition on the Fisher information matrix of all the estimated parameters, but uses the conditional Fisher information matrix restricted to the output weights only. This decreases the dimensionality of the matrix and produces more readily interpretable results.

5.3.3 Algorithm

The algorithm constitutes the following basic steps:

1. **Initialization:**

   Train the artificial neural network with $H = H^0$ (a large number of) hidden nodes.

   Go to Step 2.

2. **Design a training session** in the following two stages:

   (a) Stage I: Perform a singular value decomposition of the conditional Fisher information matrix to determine the sets K_q containing the

possible redundant nodes deemed to be making only a small contribution to the artificial neural network.

(b) Stage II: Test hypothesis (5.18) by the likelihood-ratio statistic

$$L_1 = \left(\frac{SSE_R - SSE_F}{df_R - df_F}\right) \Big/ \left(\frac{SSE_F}{df_F}\right) \tag{5.25}$$

where $df_F = IP - H$ and SSE_F are the number of degrees of freedom and the corresponding sum of squared errors (SSE) obtained for the training session respectively;

$df_R = IP - (H - \triangle H_q)$ is the number of degrees of freedom under hypothesis H_{0q} where $\triangle H_q$ is the number of weights in the set K_q, and

$$SSE_R = \min_{\mathbf{w}_{H-\triangle H_q}} \sum_p (t_p - o(\mathbf{x}_p, \widehat{\mathbf{v}}_H, \mathbf{w}_{H-\triangle H_q}))^2. \tag{5.26}$$

If $L_1 \leq F_{\alpha;df_R-df_F;df_F}$, do not reject the hypothesis H_{0q}, i.ethe reduced artificial neural network is adopted.

Use (5.25) to search over all sets K_q, $q = 1, ..., Q$ to find the maximum number of nodes that can be eliminated. Denote this number by ΔH^*.

If $\Delta H^* \geq \Delta H_{crit}$ go to Step 3.

If all of the hypotheses H_{0q}, $q = 1, .., Q$ are rejected, i.e. $\Delta H^* = 0$, or $\Delta H^* < \Delta H_{crit}$, the algorithm is stopped and the artificial neural network has H nodes.

The stopping rule ΔH_{crit} is specified by the researcher, and is determined by the complexity of the model. Setting ΔH_{crit} =1 will result in at least one node being pruned.

3. **Experiment:**

   Train the artificial neural network with $H - \Delta H^*$ nodes.

4. **Analysis** where two successive training sessions are compared using the likelihood-ratio test statistic

$$L_2 = \left(\frac{SSE_R - SSE_F}{df_R - df_F}\right) \Big/ \left(\frac{SSE_F}{df_F}\right) \tag{5.27}$$

   where df_F and df_R are the degrees of the freedom corresponding to the artificial neural networks, and SSE_F and SSE_R are the corresponding sums of squared residuals.

   If $L_2 \leq F_{\alpha;df_R-df_F;df_F}$, the reduced model with $H - \Delta H^*$ nodes is confirmed by the experiment. Go to Step 2 to further decrease the number of nodes.

   If $L_2 > F_{\alpha;df_R-df_F;df_F}$, the reduced model with $H - \Delta H^*$ nodes is rejected. Return to Step 2 to reassess the design of the training session. The number of nodes in this case in assumed to be approximately equal to $(\Delta H_{crit} + \Delta H^*)/2$.

The likelihood ratios (5.25) and (5.27) are essentially different as the SSE used in (5.25) is calculated by varying only the output weights and is therefore easy to calculate, while the SSE in (5.27) involves a training session, i.e. the calculation of all output and input weights.

5.4 Implementation and simulation

The algorithm was implemented using the Neural Network Toolbox of Matlab version 5.1.

A simulation experiment using artificial neural network regression models of which the architecture is known (i.e. the number of nodes and the weights) illustrates that the algorithm is quite efficient and requires only a moderate number of training sessions.

Simple artificial neural network regression models with $H^* = 5$ hidden nodes, initialized with $H^0 = 30$ nodes, reached their goals within 5 to 8 training sessions with an accuracy of between 1 and 3 nodes. Testing at the $\alpha = 0,05$ significance level gave the most accurate results while results with $\alpha = 0,1$ and $\alpha = 0,01$ resulted in oversmoothing and undersmoothing respectively.

As a particular result, consider the data presented in Figure 5-2. The 500 input patterns are plotted on the X-axis against the corresponding target values on the Y-Axis. Noise was generated from a $N(0,(0,05)^2)$ distribution and added to the target values. These targets with the additive noise are also plotted on the Y-axis of Figure 5-2. This vector association problem corresponds to an artificial neural network regression with $H^* = 5$ and $P = 500$. Artificial neural networks were trained with $H^0 = 30$ and $\Delta H_{crit} = 1$.

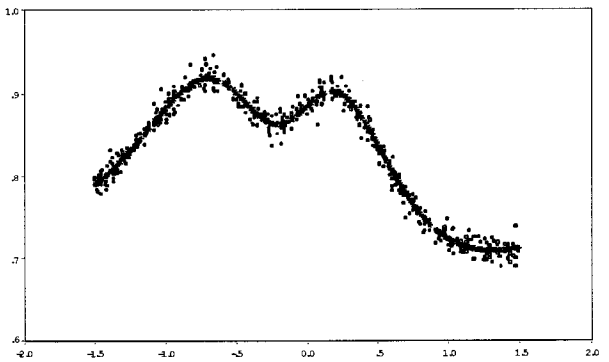

Figure 5-2: The vector association problem

Figures 5-3 to 5-5 display the results of the artificial neural networks obtained from the implementation of the algorithm for $\alpha = 0,1$, $\alpha = 0,05$ and $\alpha = 0,01$ respectively, with the input values plotted on the X-axis against the corresponding artificial neural network output values on the Y-axis.

In all cases the number of training sessions is very small compared to the 25 training sessions that would have been necessary if nodes were eliminated successively one by one. With $\alpha = 0,05$ the accurate artificial neural network model with $H = H^* = 5$ hidden nodes was obtained within 6 training sessions. Setting $\alpha = 0,01$ required 8 training sessions and resulting in an artificial neural network with $H = 3$ nodes, while 5 training sessions were required for $\alpha = 0,1$, resulting in an artificial neural network with $H = 8$ nodes.

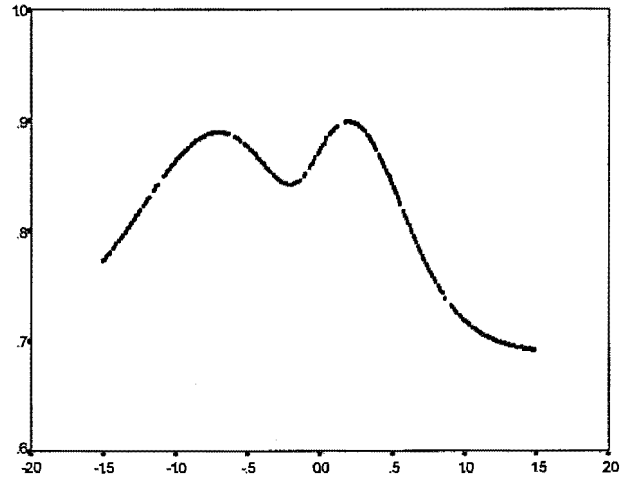

Figure 5-3: Artificial neural network using algortihm with $\alpha = 0,1$

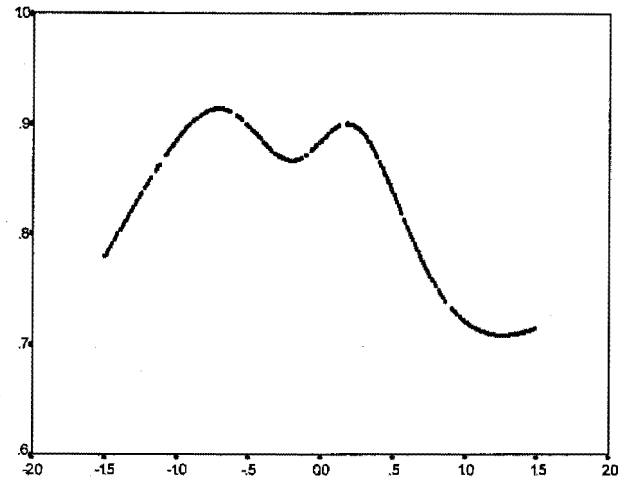

Figure 5-4: Artificial neural network using algorithm with $\alpha = 0,05$

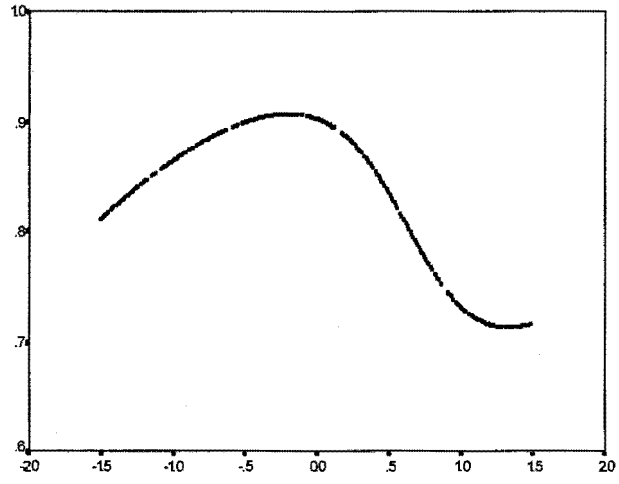

Figure 5-5: Artificial neural network using algorithm with $\alpha = 0,01$

Chapter 6

Modelling the NPRP data

The National Precipitation Research Programme (NPRP) was an exploratory weather modification experiment conducted by the South African Weather Bureau and CloudQuest, a private company, in South Africa during the summer rainfall seasons from October 1991 to March 1996. The operations took place around Bethlehem in the Free State and Carolina in Mpumalanga. Hygroscopic flares were used to seed the bases of convective storms in an attempt to enhance rainfall. The aim of the NPRP, which was a randomized experiment, was to scientifically evaluate the seeding effect on the amount of rain produced by the storms using radar measurements.

A comprehensive discussion of the experiment and the results are given in [Fletcher & Steffens, 1996] and [Mather & Fletcher, 1997]. Analyses of the results indicated that the mean and median radar-measured rain mass of seeded storms were significantly higher than that of unseeded storms approximately half an hour to an hour after the seeding decision was taken.

A study of the time history of individual storms, however, highlighted the phenomenon that not all seeded storms had responded positively to seeding as some of these storms produced very little rain after seeding.

Figure 6-1 shows an example of two seeded storms, both from the Carolina area, which had apparently reacted differently to seeding.

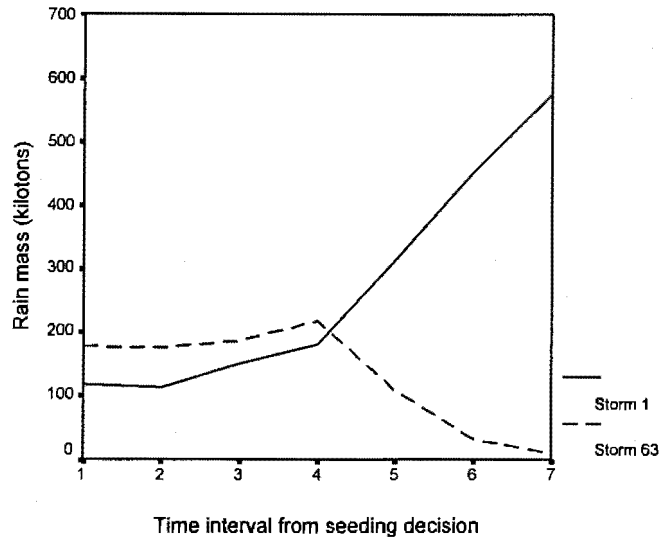

Figure 6-1: Time histories of the rain mass at lowest scan of two storms which reacted differently to seeding

Storms which reacted positively to seeding will typically resemble storm 1 while the majority of the unseeded storms resembled storm 63 in Figure 6-1, although a few of the unseeded storms also resembled storm 1.

In an attempt to distinguish between seeded storms which responded positively to seeding and storms which did not, the following proposal was made [Steffens, 1999]:

- perform a discriminant analysis using all the storms to objectively classify them into two categories, seeded and unseeded, based on the evolution of their rain mass over time;

- thereafter divide the seeded storms into those that resemble unseeded storms and those that seemed to have responded positively to seeding, using the results of the discriminant analysis;
- lastly, determine if there are any significant differences between these two groups with respect to the various radar-derived variables which are measured before seeding took place.

Being able to identify the variables that characterize storms which reacted positively to seeding will aid in better selection of storms for seeding. This will increase the probability of a positive seeding effect in operational applications such as the program which was run in South Africa's Limpopo Province (then Northern Province) after the Provincial Government approached the South African Weather Bureau, the Water Research Commission and the Department of Water Affairs and Forestry to employ cloud seeding at short notice as an emergency response to drought in that province early in 1995.

In this chapter, the NPRP data are analysed elaborating on Steffens's proposal as guidelines. Data analysts not trained in Statistics are often daunted by the different statistical techniques and their underlying assumptions. Even though this data set is not ideal for analyses using artificial neural network methodology, by virtue of the small sample size (only 127 storms of which 62 were seeded and 65 unseeded), an artificial neural network model was developed and the statistical results of the discriminant analysis compared with the output obtained by the artificial neural network model.

The preliminary results, reported at the Seventh WMO Scientific Conference on Weather Modification in 1999, elicited such an enthusiastic response from the meteorologists that it prompted the investigation as presented in this chapter. The final results were presented at a local conference where statisticians were invited to discuss "Some Statistical Problems in Industry and Science" as well as at the C. Warren Neel Conference on

The New Frontiers of Statistical Data Mining, Knowledge Discovery, and E-Business Intelligence in Knoxville, Tennessee ([Fletcher & Steffens, 2002a], [Steffens & Fletcher, 2002]). The focus in the latter paper was on the methodology of the research problem. Another paper has been accepted to be read at the 2002 Conference of the South African Society for Atmospheric Sciences [Fletcher & Steffens, 2002b]. This is the forum where the results will be communicated to the meteorologists.

The chapter is set out as follows: following the auditing of the data set, the statistical discriminant analyses were performed to obtain the objective classification of the storms into the seeded and unseeded groups. SPSS Neural Connection was subsequently used to build an artificial neural network model to classify the storms into the two groups. The results of the artificial neural network model are then compared with those of the discriminant analysis to identify the storms which were classified as seeded or unseeded by both models. Lastly, using the classification results to divide the seeded storms into those storms that resemble unseeded storms and those storms that seemed to have responded positively to seeding, analyses of variance were performed to identify possible radar-derived variables that may differentiate between the two groups.

6.1 The data set

The data file contains information on 127 storms of which 62 were seeded and 65 were not. The first seven variables relate to the identification of each storm, i.e. the date (year, month, day), the area (Carolina or Bethlehem), a radar track number, an envelope number (for the randomization process of seeding or not seeding) and a code identifying whether a storm has indeed been seeded or not. The next fourteen variables contain the radar derived rain mass for

each storm in ten minute intervals starting at ten minutes prior to the seeding decision until an hour afterwards. The seven rain masses were calculated at the lowest scan of the radar (variables $rml1$; $rml2$; ... ; $rml7$) as well as at an altitude of 6 km (variables $rmc1$; $rmc2$; ... ; $rmc7$). The next two hundred and fifty variables ($v23$, $v24$, ... , $v272$) are properties of the storms, computed from radar measurements taken in the ten minutes before the seeding decision was made. These properties are listed in Appendix A.

6.2 Data auditing

The first step, which is common to both statistical and artificial neural network modelling, is the data preparation stage which involves the auditing and preprocessing of the data as discussed in Section 4.1.

Missing values and zeroes are problems commonly encountered in radar based weather modification data. Even though many of the statistical techniques, including discriminant analysis, can be performed on data containing missing values, artificial neural networks require estimates for these missing values. SPSS Neural Connection, for example, will substitute missing values with the arithmetic mean of that variable - a technique not suitable for this particular data set.

The storms listed in Table 6.1 were identified as severely problematic as they contain either mostly zeroes or too many missing values. These 13 storms were deleted from the data set because they contain no discriminatory or predictive information.

Storm	Problem	Area	Seed
3	Non-null observations only for $rml4, rml5$	Nelspruit	No
13	$v23 - v272$ all zero values	Bethlehem	No
28	$v23 - v272$ all missing values	Nelspruit	Yes
33	Non-null observations only for $rmc1 - rmc4$	Bethlehem	Yes
52	Missing values for $rml3 - rml7$ and $rmc3 - rmc7$	Nelspruit	No
53	Missing values for $rml4 - rml7$ and $rmc4 - rmc7$	Nelspruit	Yes
54	Non-null observations only for $rml3 - rml6$	Nelspruit	No
86	$rml1 - rmc7$ either zero or close to zero; $v23 - v272$ all zeroes	Nelspruit	No
118	Missing values for $rml3 - rml7$ and $rmc3 - rmc7$	Bethlehem	No
122	Non-null observations only for $rml2 - rml7$ and $rmc2 - rmc7$	Bethlehem	No
123	Non-null observations only for $rml4 - rml7$ and for $rmc3 - rmc7$	Bethlehem	Yes
124	Non-null observations only for $rml2 - rml7$ and for $rmc2 - rmc7$	Bethlehem	No
127	Non-null observations only for $rml4 - rml7$ and for $rmc4 - rmc7$	Bethlehem	No

Table 6.1: Storms with more missing values and zeroes than observations

A further 5 storms also contained missing values for some of the variables. These storms are listed in Table 6.2. As most of the information on each storm is available for these 5 cases, the missing values were substituted by extrapolating either a quadratic or cubic regression line to the available points rather than deleting the cases. This decision conforms to the rule of thumb, used in artificial neural network modelling, that a data field is useful if at least 70% of the records contain values, i.e. 10 or more of the 14 discriminatory values for this data set.

Storm	**Missing values**	**Area**	**Seed**
16	$rml7; rmc7$	Nelspruit	Yes
48	$rml6; rml7; rmc6; rmc7$	Bethlehem	Yes
59	$rml7; rmc7$	Bethlehem	No
100	$rml7; rmc7$	Bethlehem	Yes
108	$rml6; rml7; rmc6; rmc7$	Bethlehem	No

Table 6.2: Storms with only a few missing values

The ten figures in Appendix B, Section B.2, graphically display the available data points for the storms listed in Table 6.2, with the quadratic or cubic regression lines superimposed upon them. The last two columns of Table B.1 in Appendix B, Section B.1 list the substitute values for the missing values at the lowest scan (rml) and at the 6 km scan (rmc) for each storm.

All further analyses were conducted on these remaining 114 cases with their corresponding substituted missing values. Of the original 62 seeded storms, 56 remained in the data set while 58 of the 65 original unseeded storms were included.

6.3 Discriminant analyses

Discriminant analysis is a statistical classification technique that uses linear combinations of a set of metric explanatory variables to discriminate between two or more distinct groups as defined by the categorical dependent variable. It does this by testing the hypothesis that the group means of the set of explanatory variables for two or more groups are equal. These groups should be known before the analysis is performed, and should also be non-overlapping and exhaustive. Stepwise discriminant analysis determines which exploratory variables have sufficient discriminatory power in the model [Hair & Black, 1998].

As with most parametric statistical techniques, a number of assumptions regarding the nature of the data are made, including the assumption that the data represent a sample from a multivariate normal distribution and the assumption of homoscedasticity, i.e. homogeneous variance-covariance matrices of the explanatory variables across the groups defined by the dependent variable. Discriminant analysis is robust to both these assumptions. The validity of the discriminant analysis results are more seriously affected by the presence of extreme outliers and ill-conditioned matrices, resulting from completely redundant explanatory variables. A number of diagnostics and statistical tests of assumptions are available to examine the data for violation of these assumptions. Simple histogram analyses, often employed in artificial neural network modelling, show that most of the variables are positively skewed and leptokurtic. Nonetheless, the analyses were performed on the untransformed data, mainly because of the problems encountered with the numerous zero values.

Three different analyses were conducted on the data set to obtain the statistical classification of all the storms into the two groups: seeded and unseeded (i.e. those storms which resembled seeded storms based on their rain mass evolution, and those which did not). Firstly all possible discriminating variables $rml1$; ... ; $rml7$; $rmc1$; ... ; $rmc7$ were entered into the model in a

single step. Secondly the same 14 variables were used in a stepwise procedure using SPSS's default values for entering and removing discriminating variables (5% and 10% respectively). Lastly the stepwise procedure was repeated by relaxing the criteria for entering and removing variables from the model to 10% and 15% respectively.

As discriminant analysis can be performed on data containing missing values, all three analyses were repeated on the data set containing the missing values instead of the substitute values. The results were practically identical, indicating that the substituted values did not influence the results of the discriminant analysis.

The SPSS syntax files and complete output is stored as Appendix C in pdf format on the CD which is included at the back.

6.3.1 Enter: $rml1$; ... ; $rml7$; $rmc1$; ... ; $rmc7$

One of the statistical tests available in SPSS as a diagnostic is Box's M test which tests the null hypothesis of homoscedasticity. The null hypothesis was rejected for this analysis (and the other discriminant analyses), indicating that the assumption of equal population covariance matrices is violated. As mentioned above, discriminant analysis is robust to this assumption, and the analyses were continued.

The results of the discriminant analysis where all 14 possible explanatory variables $rml1$; ... ; $rml7$; $rmc1$; ... ; $rmc7$ were entered in a single step are displayed in Table 6.3. The row percentages are also displayed in this table as they depict the discriminant analysis classifications across the observed group membership (seeded and not seeded). Overall, 74 of the 114 (i.e. 65%) of the original storms were correctly classified as either seeded or unseeded. The 40 remaining storms (35%) displayed rain mass evolutions that were inconsistent with their individual classifications as the discriminant analysis model could

not find significant differences between the mean rain mass of the 30 seeded storms that were classified as unseeded in Table 6.3 and the 46 unseeded storms that were correctly classified by the model, nor between the mean rain mass of the 10 unseeded storms that were classified as seeded in Table 6.3 and the 28 unseeded storms that were correctly classified by the model.

		Predicted group membership		
		Not seeded	Seeded	**Total**
Observed group membership	Not seeded	46 (82%)	10 (18%)	56 (100%)
	Seeded	30 (52%)	28 (48%)	58 (100%)

Table 6.3: Discriminant analysis classification entering all 14 discriminating variables $rml1; \ldots; rml7; rmc1; \ldots; rmc7$

6.3.2 Stepwise: $rml1; \ldots; rml7; rmc1; \ldots; rmc7$ at 5% to enter and 10% to exit (SPSS default)

Following the discriminant analysis where all the explanatory variables were forced into the model, a stepwise discriminant analysis was performed simply using SPSS's default probabilities to enter the model and to be removed from the model (15% and 10% respectively).

For this stepwise discriminant analysis only the rain mass at the lowest scan 40 - 50 minutes after the seeding decision took place, $rml6$, proved to have significant discriminatory power between the seeded and unseeded storms.

With only one discriminatory variable, fewer of the original storms were correctly classified (67 storms, i.e. 59%) as opposed to forcing the use of all possible discriminating variables (65%). Table 6.4 displays the results of the discriminant analysis. Of the seven additional storms that could not be identified as belonging to their original classification in this analysis, three were unseeded and four seeded.

		Predicted group membership		
		Not seeded	Seeded	**Total**
Observed group membership	Not seeded	43 (77%)	13 (23%)	56 (100%)
	Seeded	34 (59%)	24 (41%)	58 (100%)

Table 6.4: Discriminant analysis classification using a stepwise procedure with SPSS default values

6.3.3 Stepwise procedure with relaxed criteria

After inspection of the stepwise discriminant analysis results it was decided to repeat the stepwise procedure, but to relax the criteria for entering and removing variables from the model to 10% and 15% respectively.

With the relaxed criteria for variables to enter or exit the model, both *rml5* and *rml6*, the rain mass at the lowest scan between 30 to 40 minutes and the rain mass at the lowest scan between 40 to 50 minutes after the seeding decision took place, were found to have significant discriminatory power. (The

rain mass at the 6 km scan 30 - 40 minutes after the seeding decision took place, $rmc5$, had a significance value to enter the model equal to 0,109 at this step, but was completely redundant in the next step. This no doubt reflects the inherent collinearity of the two radar measurements at the lowest scan and at the 6 km scan during the same time period.) By including $rml5$ in the model the three unseeded storm which were misclassified in the previous model were again correctly classified, resulting in 61% correct classifications for this model. The results are displayed in Table 6.5.

		Predicted group membership		
		Not seeded	Seeded	**Total**
Observed group membership	Not seeded	46 (82%)	10 (18%)	56 (100%)
	Seeded	34 (59%)	24 (41%)	58 (100%)

Table 6.5: Discriminant analysis classification using a stepwise procedure with relaxed conditions

The problem of seeded storms behaving like unseeded storms, i.e. displaying a rain mass evolution similar to the unseeded storms, as explained at the beginning of the chapter, becomes clear on inspection of Tables 6.3 - 6.5.

For the last analysis, only 24 of the 58 seeded storms could be correctly classified as seeded using discriminant analysis to classify the storms based on their mean rain mass. The majority of the seeded storms (59%) could not be distinguished from the unseeded storms by the discriminant functions,

suggesting that seeding these storms was a useless exercise as they evidently did not react to the seeding.

Furthermore, of the 56 unseeded storms, 10 were classified as seeded using discriminant analysis, i.e. 18% of the unseeded storms displayed properties based on the rain mass evolution over time that were similar to the seeded storms which were also classified as seeded based on the discriminant function. Inferring that 18% of the seeded storms would in all likelihood also have displayed properties similar to seeded storms regardless of seeding status leaves only 14 storms that reacted to seeding - less than a quarter of the 58 storms that were seeded during the experiment.

The importance of the ability to identify appropriate storms for seeding based on information available prior to the seeding decision in an operational seeding program is obvious.

It was decided to use the statistical classification of seeded and unseeded storms obtained by the stepwise procedure with the relaxed criteria (Section 6.3.3) in the further analyses which attempted to determine possible predictors to identify storms that are likely to respond positively to seeding.

The linear discriminant equation for this analysis is

$$D = -0,005rml5 + 0,006rml6 - 0,37.$$

The function is very successful in separating the storms into two groups based on their rain mass evolutions, as evidenced by inspection of the means for the two different groups. Of particular interest is the discriminatory power of this function for the storms which were seeded during the experiment. The number of storms, the means and corresponding standard deviations are displayed in Table 6.6 for all 114 storms as well as for the 58 seeded storms.

		Number	Mean	Std dev.
All storms	"Unseeded"	80	-0,4423	0,2393
	"Seeded"	34	1,0408	1,3942
Seeded storms	"Unseeded"	34	-0,4179	0,2792
	"Seeded"	24	1,2650	1,5525

Table 6.6: Discriminant analysis classification statistics

6.4 Artificial neural network classification

An artificial neural network model was also developed to classify storms as seeded or unseeded based on the information contained in the rain mass data. The aim was to compare the artificial neural network storm classification with the classification obtained by the discriminant analysis. As mentioned before, this data set is very small in terms of artificial neural network applications.

Figure 6-2 displays the SPSS Neural Connection workspace where the topology is assembled. The topology consists of a data input tool, connected to a modelling tool which is in turn connected to two output tools.

In this application the NPRP data file is imported as record delimited data. The data input tool, where the data are displayed as a spreadsheet, allows one to specify the format of the data, to choose the training, test and validation set sizes and to set the uses of the individual fields in the data as input fields, target fields, reference fields or unused fields. Reference fields are passed through the application and is displayed on output, but are not used or changed by the artificial neural network. Two methods are available for ordering the records within the data set: sequential and random.

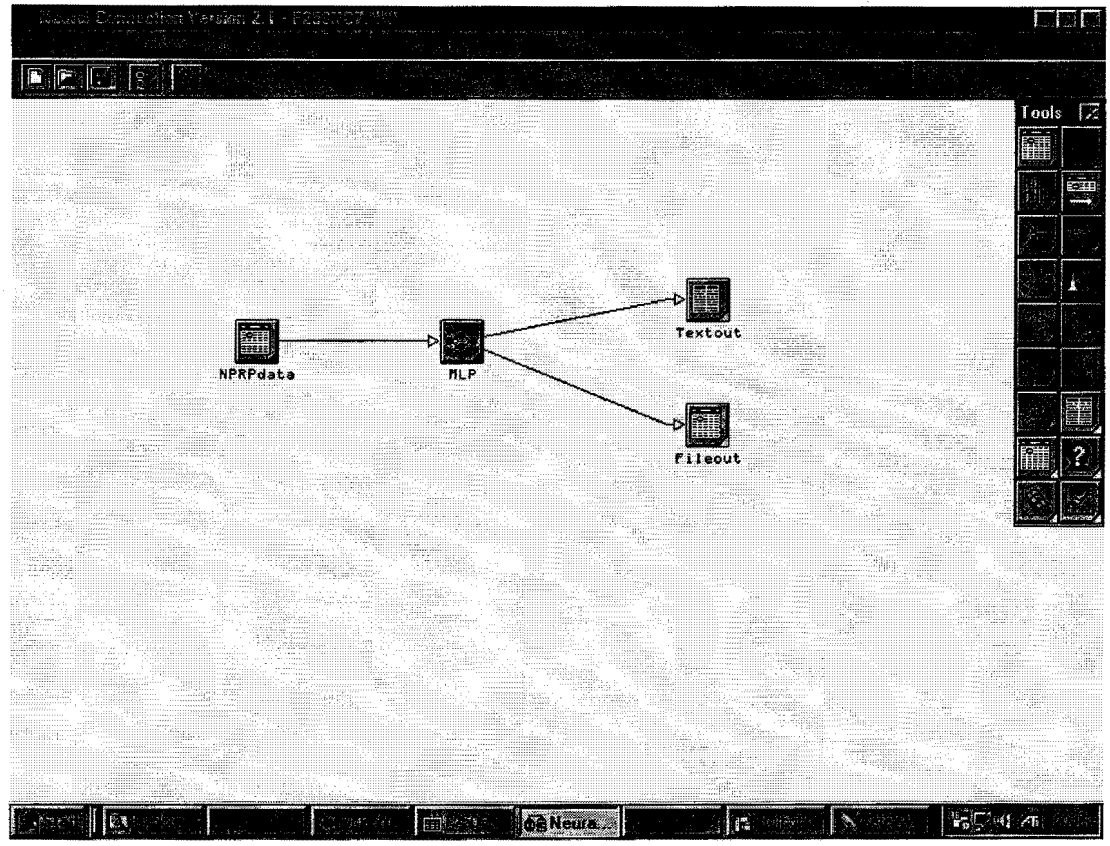

Figure 6-2: SPSS Neural Connection topology

Knowing that a seeding effect can only be detected approximately 20 minutes after seeding took place, the input variables were specified as the rain mass at the last four time intervals for the radar scans at cloud base and at 6 km altitude. The target variable is the seeding status of the storm. (The first three time windows correspond to the 10 minutes before seeding and the 0 - 10 and 10 - 20 minute intervals afterwards and were flagged as unused fields.)

SPSS Neural Connection automatically allocates 80% of the records to the training set, and 10% to the validation and test sets respectively.

The purpose of this artificial neural network model, however, is simply to classify the storms as seeded or unseeded based on their rain mass. The model is not intended to be used for forecasting or prediction. The sample was

therefore divided into training and validation subsets only, using the proposal by Smith that two-thirds of the sample is used for training and one third for validation [Smith, 1993]. The validation set was also used to test the artificial neural network's performance. Because the records are date ordered, the default assignment of records to the training and validation sets was changed from sequential to random.

The multilayer perceptron (MLP) model in SPSS Neural Connection was chosen as the artificial neural network modelling tool for this classification problem. This feedforward artificial neural network, which has one or more layers of nodes between the input and output nodes, is one of the most commonly used artificial neural network models in practice. The model employs the backpropagation learning algorithm discussed in Section 4.5.

SPSS Neural Connection has the option to either use the data as presented or to standardize either the input data or the output data or both.

The nodes in the hidden layer are automatically generated by default, although the number of hidden layers and nodes can be specified by the user.

There are three options available for the activation functions in the hidden layers, i.e. linear, tanh (the default) or sigmoid (cf. Section 4.3).

The default weight distribution is uniform, but can be set to Gaussian. The parameters for both these distributions can be manually adjusted.

Two methods are available for the backpropagation of the error: steepest descent and conjugate gradient. The conjugate gradient method is the default in SPSS Neural Connection. This method differs from the method of steepest descent, where the gradient of the error surface is measured after each pass and the weights simply changed in the direction of the steepest gradient, as the weights are altered using a compromise between the direction of the steepest gradient and the previous direction of change ([SPSS, 1995], [Fine, 1999], [Haykin, 1999]).

The output tools consist of a text output tool which runs topologies and displays the results and success rate in text format to the screen. The text output tool can also write files in ASCII or SPSS file format. The data output tool allows one to view the data set that has been passed through the application and to examine the results as well as saving the results in a named file. The fields to be written out can be specified by the user.

The multilayer perceptron that was generated automatically by SPSS Neural Connection has eight input nodes, one hidden layer with three hidden nodes and an output node. The weights are included as part of the complete output from the artificial neural network which is stored as Appendix D in pdf format on the enclosed CD.

Figure 6-3 displays the $8-3-1$ MLP architecture for the NPRP data.

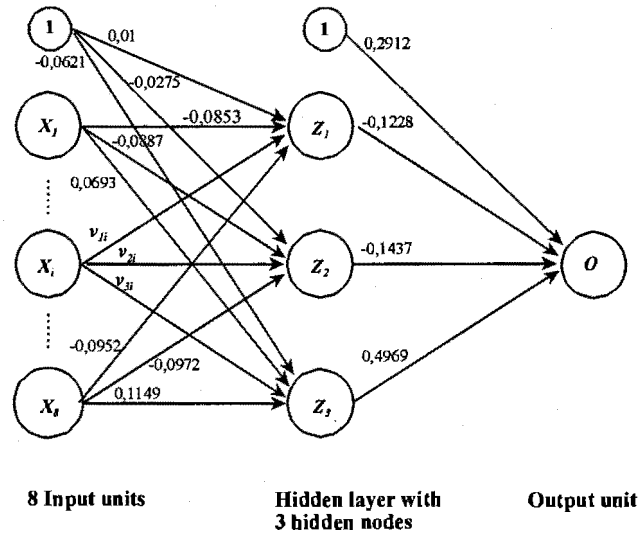

Figure 6-3: Multilayer perceptron (MLP) architecture for the NPRP data

Tables 6.7 and 6.8 display the results of the artificial neural network model for the training and validation (i.e. test) sets respectively, together with the row percentages depicting the artificial neural network model classification across the two groups.

		Predicted group membership		
		Not seeded	Seeded	Total
Observed group membership	Not seeded	30 (75%)	10 (25%)	40 (100%)
	Seeded	11 (31%)	25 (69%)	36 (100%)

Table 6.7: Artificial neural network model results for the training set

		Predicted group membership		
		Not seeded	Seeded	**Total**
Observed group membership	Not seeded	12 (25%)	4 (75%)	16 (100%)
	Seeded	14 (64%)	8 (36%)	22 (100%)

Table 6.8: Artificial neural network model results using the validation set

In total, 39 storms were misclassified by this model, of which 25 were unseeded. The percentage of seeded storms misclassified by the artificial neural network model as unseeded (43% overall) is much lower than the percentage of seeded storms misclassified by the discriminant analysis (59%) discussed in Section 6.3.3. However, the corresponding percentage of the validation (or test) set by itself is 64%, indicating that model performance may be comparable.

(It is of course possible to increase the artificial neural network model's performance either by increasing the number of nodes in the hidden layer or by allowing the model to overtrain by continuing the number of iterations. The aim here is, however, not to obtain a model that fits the data nearly perfectly, but rather to obtain a model that will distinguish between the inherent characteristics of the two types of seeded storms: those that resemble seeded storms and those that do not.)

The artificial neural network results are summarized in Table 6.9.

		Predicted group membership		
		Not seeded	Seeded	**Total**
Observed group membership	Not seeded	42 (75%)	14 (25%)	56 (100%)
	Seeded	25 (43%)	33 (57%)	58 (100%)

Table 6.9: Artificial neural network model results

As was the case in the discriminant analysis, the phenomenon of seeded storms having the same properties as unseeded storms with respect to rain mass evolution over time is evident on inspection of the table. Just more than half of the seeded storms (33 of the 58, i.e. 57%) could be correctly classified as seeded by the artificial neural network model. The remaining storms resembled the unseeded storms and could thus not be distinguished by the model as seeded. This corroborates the discriminant analysis evidence that seeding these storms was futile. Furthermore, 14 of the 56 unseeded storms were classified as seeded by the artificial neural network model, indicating that a quarter of the unseeded storms were in some way similar to the seeded storms which were classified as seeded by the model. (The corresponding figure for the discriminant analysis was 18%.)

Again reasoning that, based on this model, 25% of the seeded storms would in any case have displayed properties similar to seeded storms regardless of seeding status only 32% of the storms, i.e. 18 or 19 storms, remain that actually reacted positively to seeding. This agrees with the findings of the discriminant analysis, namely that it is extremely important to identify storms that are suitable for seeding.

6.5 Model comparison

The performance of the two models with respect to separating storms is assessed in this section. Discriminant analysis is a popular and efficient statistical technique that is widely used by statistically literate data analysts. Artificial neural networks, however, are becoming increasingly popular with data analysts who do not have a statistical background. The so-called "blackbox" approach that is often followed when implementing an artificial neural network model appeals to many users who feel uncomfortable with their lack of

knowledge of Statistics and the accompanying assumptions. When faced with large data sets, resorting to an artificial neural network is especially attractive to many data analysts as its effectiveness as a modelling tool has been widely proved (cf. Section 3.10). The challenge with the NPRP data set was to see how well an artificial neural network model performs on smaller data sets, as the methodology is so readily available and is bound to be implemented by users who have access to it.

The overall percentage of seeded and unseeded storms correctly classified by the artificial neural network model (75 storms, i.e. 66%) is slightly better than the result obtained by the final discriminant analysis model (70 storms, i.e. 61%).

For the artificial neural network model, 33 of the seeded and 42 of the unseeded storms were correctly identified (cf. Table 6.9), while the discriminant model correctly classified 24 storms as seeded and 46 storms as unseeded (cf. Table 6.5).

The two models have 22 storms in common which were correctly classified by both as seeded. This implies that the seeded storms which were correctly classified by the discriminant analysis model is basically a subset of the correctly classified artificial neural network model storms. As far as the unseeded storms which were correctly classified as unseeded is concerned, the two models had 40 storms in common.

Table 6.10 displays the crosstabulation of the discriminant analysis and artificial neural network classifications with the cell percentages in brackets underneath each count. The two models have an 82% correspondence (93 out of 114 cases).

		Neural network		
		Not seeded	Seeded	**Total**
Discriminant analysis	Not seeded	63 (55%)	17 (15%)	80 (70%)
	Seeded	4 (4%)	30 (26%)	34 (30%)
	Total	67 (59%)	47 (41%)	114 100%)

Table 6.10: Crosstabulation of discriminant analysis by artificial neural network classification

A comparison of the graphs of the means of all the seeded and unseeded storms and the graphs of the seeded storms classified as "seeded" and "unseeded" by the discriminant analysis model and the artificial neural network model respectively clearly indicates that both models perform more than adequately in separating storms which had rain mass evolutions consistent with those of seeded storms and those storms which did not.

Figures 6-4 to 6-9 display these rain mass graphs at both the lowest radar scan and the scan at an altitude of 6 km.

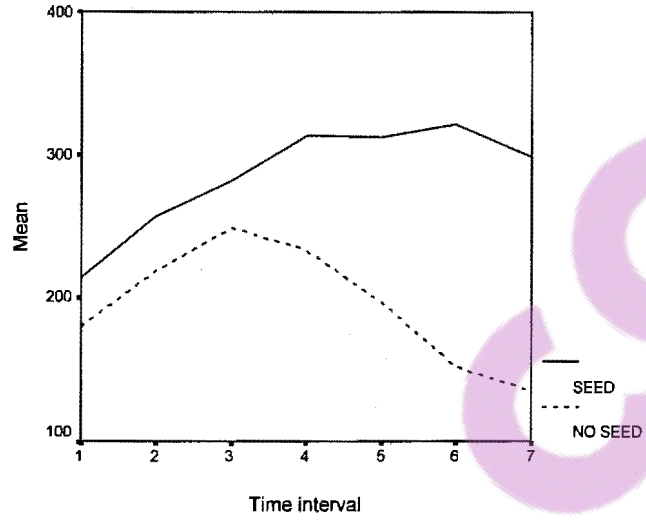

Figure 6-4: Rain mass means of the seeded and unseeded storms at the lowest radar scan

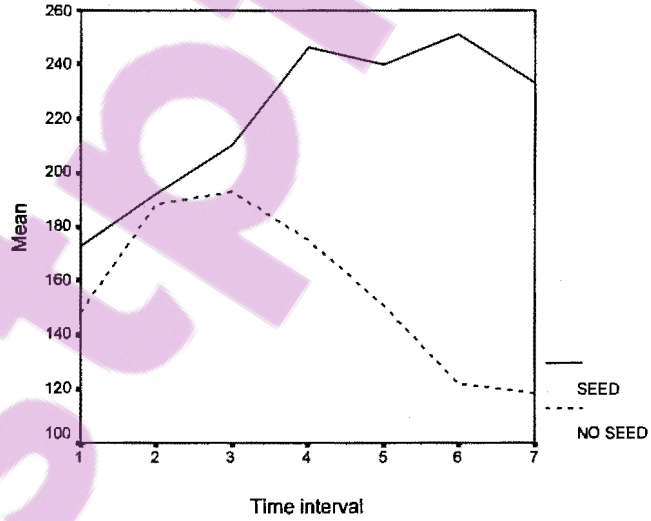

Figure 6-5: Rain mass means of the seeded and unseeded storms at the 6km radar scan

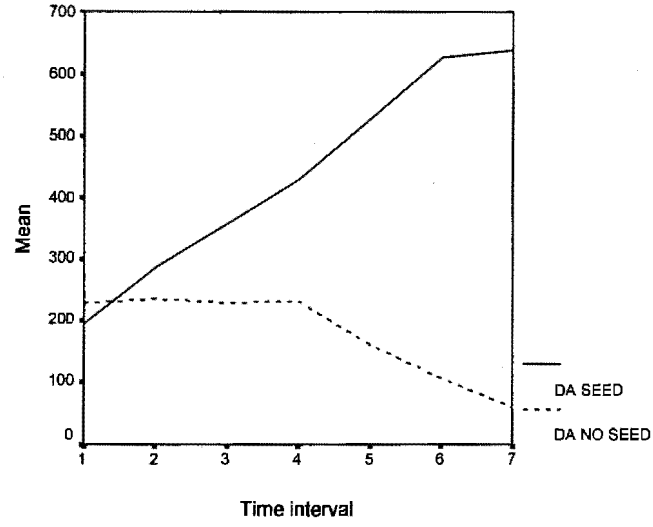

Figure 6-6: Rain mass means of the seeded storms at the lowest radar scan, classified as "seeded" and "unseeded" by the discriminant analysis

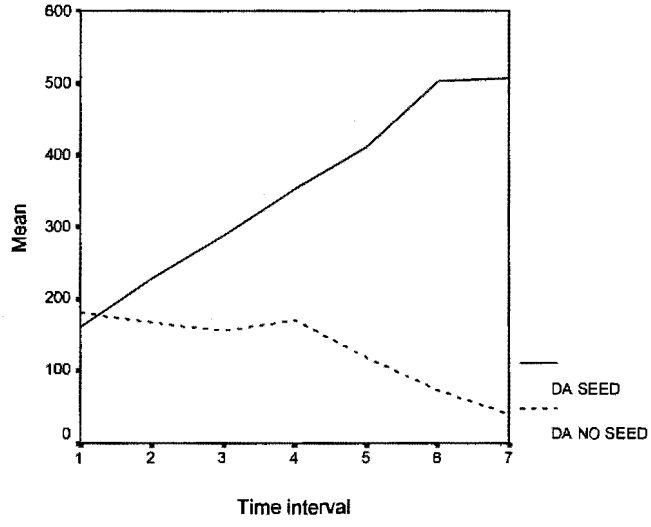

Figure 6-7: Rain mass means of the seeded storms at the 6 km radar scan, classified as "seeded" and "unseeded" by the discriminant analysis

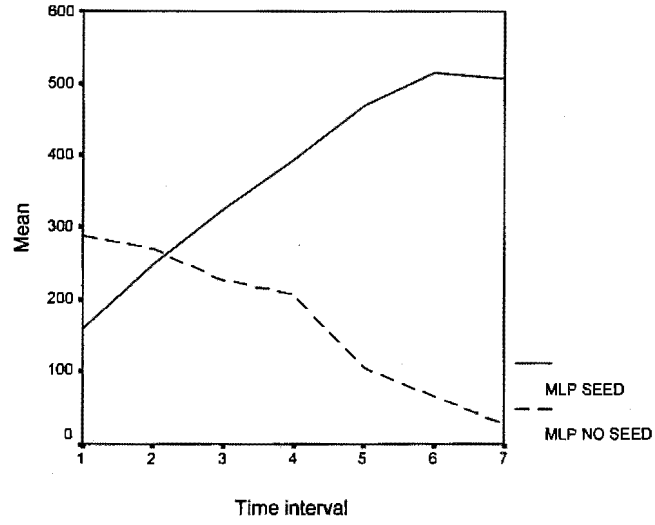

Figure 6-8: Rain mass means of the seeded storms at the lowest radar scan, classified as "seeded" and "unseeded" by the neural network

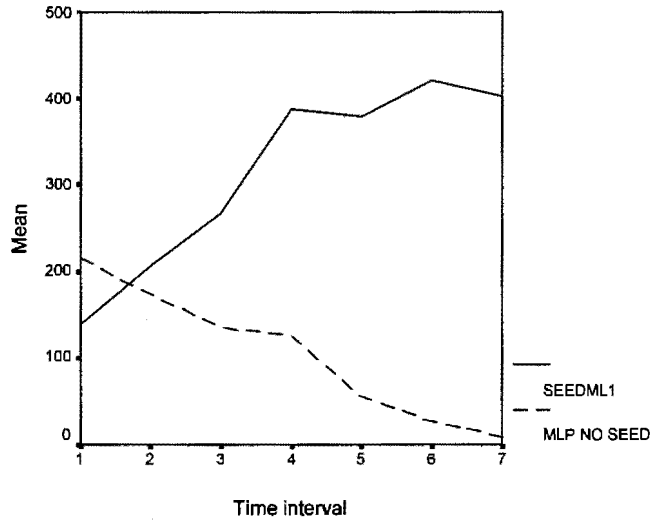

Figure 6-9: Rain mass means of the seeded storms at the 6 km radar scan, classified as "seeded" and "unseeded" by the neural network

6.6 Analysis of variance

Analysis of variance (ANOVA) is a statistical technique to test hypotheses about differences between two or more group means defined by a single metric variable [Hair & Black, 1998]. ANOVA is based on the assumptions that the explanatory variables are quantitative and normally distributed within the groups defined by a single categorical variable. Homogeneity of the variances in the different groups are also assumed. ANOVA is remarkably robust to departures from these assumptions and violations are generally tolerated well.

The aim at this stage of the study was to characterize the seeded storms in the experiment which apparently responded positively to seeding as opposed to the seeded storms which apparently did not respond to the seeding. As interest is restricted to the behaviour of seeded storms only, the 58 seeded storms were divided into two groups based on the final model classification results. In the case of the discriminant analysis model, 24 of the seeded storms were classified as seeded and 34 storms as unseeded, while 33 of the seeded storms were classified as seeded by the artificial neural network model, and 25 as unseeded. The two groups of storms, "seeded" and "not seeded", were consequently compared with respect to the means of several explanatory variables. This comparison was done for each of the two classification models.

The explanatory variables represent 250 different storm properties such as echo tops, storm depth, storm volume, storm mass, storm area, rain flux, precipitable water content and reflectivity. These storm properties were computed using the computer program TRACKPROPS ([Dixon & Mather, 1986], [Steffens, 1999]). All calculations are based on the radar measurements taken in the ten minutes before the seeding decision was taken. The importance of this fact is that any variables that are identified as having possible explanatory power may be calculated by the radar operators in real time before the actual decision to seed a particular storm is taken. The storm properties, with a

short description of each, are listed in Appendix A at the end of the thesis.

As a first step in the attempt to identify those storm properties that could possibly explain the phenomenon of seeded storms having different rain mass evolutions, simple oneway analyses of variance were performed. These analyses were purely of an exploratory nature. The analyses of variance were performed using each of the 250 radar-derived variables $v23; \ldots ; v272$ as the factor with the two model classifications of the seeded storms as the dependent variable in each case. The results must obviously be interpreted with circumspection because of the multiplicity of tests that were performed.

To assess the violation of the assumption of homoscedasticity, Levene's homogeneity-of-variances test was included with each of the analyses. (Levene's test was chosen as it is not sensitive to departures from the assumption that the data are a random sample from a normal population - which is not the case for this data set as mentioned in Section 6.3.)

Following the simple oneway analyses of variance, two analyses of variance were done - one for each of the two classification models (discriminant analysis and artificial neural network) - where all the storm properties which had a large difference between the two classification groups ("seeded" and "unseeded") in the first step were now entered in a single step. Finally, analyses of variance were performed - again one for each of the two classification models - using only those storm properties which were identified as having large differences between the means of the two classification groups in the first step and which also did not violate the homoscedasticity assumption.

The analyses of variance where multiple explanatory variables are entered, as opposed to the multiple tests where the variables are entered one by one, give more reliable results as the problem with the power of the multiplicity of tests is addressed [Hair & Black, 1998]. The SPSS syntax files and complete output is stored as Appendix E in pdf format on the enclosed CD.

6.6.1 Discriminant analysis classification

Of the 58 seeded storms, 34 were classified as unseeded and 24 as seeded by the discriminant analysis model (cf. Table 6.5). The 250 individual ANOVAs which were performed identified 35 storm properties which have large differences between the means of the discriminant analysis model defined seeded and unseeded groups. These 35 storm properties are listed in Table 6.11. Also listed are the means of these variables for the 56 unseeded and 58 seeded storms, as well as for the 58 seeded storms which were classified by the discriminant analysis as unseeded and seeded.

The significance level of the F-statistics that is used to test for the difference between the means is displayed in column two of the table. An asterisk (*) next to the variable indicates that the difference between the means is significant at the 1% level for that variable. Column three displays those variables which do not violate the assumption of homoscedasticity, based on Levene's test for homogeneity of variances. It is important to note that the 1% and 5% levels of significance which are used to detect significant differences between the group means serves here merely as an indication that large differences exist between the storms which had apparently reacted positively to seeding and those which had apparently not for the corresponding variables.

No substantial differences between the discriminant analysis defined seeded and unseeded groups were found for any of the variables where the storm properties $v32$, $v34$, $v38$, ... , $v262$, $v272$ were entered in a single step in the ANOVA. For the ANOVA where the 21 homoscedastic explanatory variables were entered as covariates (cf. column three in Table 6.11) a large difference at the 5% level between the two groups was obtained only for $v166$ (Mean dBz at 6 km: Time to max rate of increase). At the 10% level, $v67$ (Storm mass at 6 km: Time to max mass) also had a large difference between the means of the two groups.

Variable	Significance level	Levene	Discriminant analysis		True status	
			Unseeded	Seeded	Unseeded	Seeded
$v32$: Peak dBz at decision time	0,025		46,77	54,36	48,70	49,91
$v34$: Echo top: Max rate of increase	0,034		2,46	4,20	3,96	3,18
$v38$: Echo top: Persistence	0,018	✓	-0,03	0,04	0,01	-0,00
$v39$: Echo top: Max ratio	0,046		0,03	0,08	0,07	0,05
$v51^*$: Storm volume: Time to max	0,009		5,82	8,50	6,65	6,93
$v53$: Storm volume: Persistence	0,029	✓	0,09	0,26	0,12	0,16
$v59^*$: Total storm mass: Time to max	0,008		4,99	7,89	5,50	6,19
$v61$: Total storm mass: Persistence	0,047	✓	0,06	0,25	0,08	0,14
$v64$: Storm mass at 6 km: Max rate of increase	0,030		0,12	0,31	0,13	0,20
$v67^*$: Storm mass at 6 km: Time to max mass	0,001	✓	4,28	7,77	5,36	5,72
$v68$: Storm mass at 6 km: Time to max rate of increase	0,018	✓	3,25	5,60	4,77	4,22
$v69^*$: Storm mass at 6 km: Persistence	0,008	✓	-0,08	0,22	0,01	0,04
$v83$: Storm area at 6 km: Time to max	0,024		5,64	8,04	6,11	6,63
$v99^*$: Rain flux at 6 km: Time to max	0,010	✓	4,71	7,47	5,39	5,85
$v100$: Rain flux at 6 km: Time to max rate of increase	0,013	✓	3,63	6,02	5,34	4,62
$v101$: Rain flux at 6 km: Persistence	0,034	✓	0,01	0,25	0,05	0,11
$v107$: Precipitable water: Time to max	0,011		5,54	8,18	5,94	6,63
$v108$: Precipitable water: Time to max rate of increase	0,032	✓	3,51	5,68	4,94	4,40

Table 6.11: ANOVA results for the discriminant analysis classification

Variable	Significance level	Levene	Discriminant analysis		True status	
			Unseeded	Seeded	Unseeded	Seeded
$v118$: Vertical centroid: Time to max	0,036	✓	3,16	5,31	4,14	4,05
$v120$: Vertical centroid: Persistence	0,047	✓	-0,05	0,00	-0,03	-0,03
$v125$*: Reflectivity-weighted centroid: Time to max	0,010	✓	3,01	5,46	4,19	4,03
$v126$: Reflectivity-weighted centroid: Time to max rate of increase	0,049	✓	3,10	4,85	4,72	3,83
$v127$: Reflectivity-weighted centroid: Persistence	0,016	✓	-0,06	0,01	-0,03	-0,03
$v141$: Peak dBz over whole volume: Max	0,029		52,54	55,49	54,03	53,76
$v143$: Peak dBz over whole volume: Mean	0,045	✓	50,73	53,62	52,47	51,93
$v165$: Mean dBz at 6 km: Time to max	0,029	✓	3,81	6,11	4,88	4,76
$v166$*: Mean dBz at 6 km: Time to max rate of increase	0,001	✓	3,14	6,12	5,39	4,37
$v169$: Max height of 45 dBz: Max	0,034	✓	6997,90	8745,50	7827,66	7721,04
$v171$: Max height of 45 dBz: Mean	0,048	✓	6047,87	7722,91	6939,95	6740,99
$v182$: Height of peak dBz: Max ratio	0,027		0,07	0,15	0,09	0,10
$v212$: Summary statistic of dBz as a function of height	0,013	✓	1,08	0,52	1,17	0,85
$v247$: Summary statistic of the lowest scan as a function of dBz	0,031		0,00	0,00	0,00	0,00
$v252$: Summary statistic of the lowest scan as a function of dBz	0,049		0,00	0,00	0,00	0,00
$v262$: Summary statistic of the lowest scan as a function of dBz	0,034		0,01	0,01	0,01	0,01
$v272$*: Max dBz in day	0,001		58,57	61,74	60,74	59,88

Table 6.11 (ctd): ANOVA results for the discriminant analysis classification

6.6.2 Multilayer perceptron (MLP) classification

The artificial neural network model, using the multilayer perceptron, classified 33 of the 58 seeded storms as seeded and 25 storms as unseeded(cf. Tables 6.6 and 6.7).

Thirty-one storm properties were identified which have large differences between the means of the artificial neural network defined seeded and unseeded groups, based on the results of the 250 individual ANOVAs. These properties are listed in Table 6.12. The variables which do not violate the assumption of homoscedasticity, based on Levene's test for homogeneity of variances, are marked in column three of the table. The means of these variables for the 56 unseeded and 58 seeded storms, as well as for the 58 seeded storms which were classified as unseeded and seeded by the artificial neural network model are also included.

When the 31 storm properties $v32$, $v38$, $v64$, ... , $v246$, $v261$ were entered in a single step in the ANOVA, a substantial difference between the artificial neural network model defined seeded and unseeded groups were found only for $v216$ (Summary statistic of dBz as a function of height), as indicated by the 5% level of significance of the ANOVA. Two more variables, $v166$ (Mean dBz at 6 km: Time to max rate of increase) and $v222$ (Summary statistic of dBz as a function of height), had large differences between the means of the two groups at the 10% level.

Performing an ANOVA using the 17 homoscedastic explanatory variables identified four variables which have large differences between the artificial neural network model defined seeded and unseeded groups: $v166$ (Mean dBz at 6 km: Time to max rate of increase), $v169$ (Max height of 45 dBz: Max), $v216$ (Summary statistic of dBz as a function of height) and $v222$ (Summary statistic of dBz as a function of height).

Variable	Significance level	Levene	Neural network		True status	
			Unseeded	Seeded	Unseeded	Seeded
$v32$: Peak dBz at decision time	0,032		45,80	53,02	48,70	49,91
$v38$: Echo top: Persistence	0,022	✓	-0,04	0,03	0,01	-0,00
$v64$: Storm mass at 6 km: Max rate of increase	0,050		0,10	0,27	0,13	0,20
$v67$: Storm mass at 6 km: Time to max mass	0,013	✓	4,17	6,90	5,36	5,72
$v68^*$: Storm mass at 6 km: Time to max rate of increase	0,008		2,75	5,34	4,77	4,22
$v69^*$: Storm mass at 6 km: Persistence	0,006	✓	-0,13	0,17	0,01	0,04
$v70$: Storm mass at 6 km: Max ratio	0,048		0,16	0,30	0,22	0,24
$v99$: Rain flux at 6 km: Time to max	0,031	✓	4,53	6,85	5,39	5,85
$v116^*$: Vertical centroid: Max rate of increase	0,002		0,36	1,37	0,97	0,93
$v118^*$: Vertical centroid: Time to max	0,002	✓	2,33	5,36	4,14	4,05
$v120^*$: Vertical centroid: Persistence	0,001	✓	-0,07	0,01	-0,03	-0,03
$v121$: Vertical centroid: Max ratio	0,018		0,01	0,05	0,03	0,03
$v123^*$: Reflectivity-weighted centroid: Max rate of increase	0,000		0,40	1,89	1,22	1,25
$v126$: Reflectivity-weighted centroid: Time to max rate of increase	0,035	✓	2,77	4,63	4,72	3,83
$v127^*$: Reflectivity-weighted centroid: Persistence	0,003	✓	-0,09	0,00	-0,03	-0,03
$v128^*$: Reflectivity-weighted centroid: Max ratio	0,009		0,01	0,06	0,03	0,04

Table 6.12: ANOVA results for neural network model classification

Variable	Significance	Levene	Neural network		True status	
	level		Unseeded	Seeded	Unseeded	Seeded
$v144$: Peak dBz over whole volume: Time to max	0,030	✓	4,12	6,44	5,33	5,44
$v166$: Mean dBz at 6 km: Time to max rate of increase	0,015	✓	3,06	5,36	5,39	4,37
$v169$*: Max height of 45 dBz: Max	0,007	✓	6474,39	8665,48	7827,66	7721,04
$v171$: Max height of 45 dBz: Mean	0,044	✓	5775,42	7472,47	6939,95	6740,99
$v172$: Max height of 45 dBz: Time to max	0,039	✓	3,29	5,41	4,47	4,49
$v179$*: Height of peak dBz: Time to max	0,008	✓	2,89	5,73	4,51	4,51
$v182$: Height of peak dBz: Max ratio	0,045		0,06	0,13	0,09	0,10
$v186$*: Summary statistic of mass as a function of height	0,000		0,30	1,56	1,07	1,02
$v188$: Summary statistic of mass as a function of height	0,045		1592,97	1853,44	1714,11	1741,17
$v189$: Summary statistic of mass as a function of height	0,036	✓	1721,79	2003,21	1828,65	1881,91
$v190$: Summary statistic of mass as a function of height	0,044		1442,79	1712,41	1578,69	1596,19
$v216$: Summary statistic of dBz as a function of height	0,045	✓	0,00	0,00	0,00	0,00
$v222$: Summary statistic of dBz as a function of height	0,034	✓	7,66	4,51	5,30	5,87
$v246$: Summary statistic of 3° area as a function of dBz	0,041		0,02	0,00	0,01	0,01
$v261$: Summary statistic of 3° area as a function of dBz	0,024		0,03	0,01	0,02	0,02

Table 6.12 (ctd.): ANOVA results for neural network model classification

6.6.3 Model correspondence

Fifteen storm properties were found to have substantial differences between the so-called seeded and unseeded groups for both the discriminant analysis and the artificial neural network models. These properties are listed in Table 6.13 together with the means of the 58 seeded and the 56 unseeded storms, and the means of the 58 seeded storms classified as unseeded and seeded by the discriminant analysis and artificial neural network models respectively.

From this table it is seen that all the variables have higher values for the "seeded" storms than for the "unseeded" storms for both classification models. Persistence, a property for which large differences between the two groups were found for four of the variables, is the ratio of the maximum value during the 10 minutes prior to seeding to the maximum value at the beginning of the period, possibly indicating that these storms were still growing at the time of seeding. The time to reach a maximum is another property for which large differences between the two groups were found, for seven of the variables, supporting this notion. The remaining four variables for which large differences between the two groups were found all relate to dBz, i.e. the intensity of the reflectivity in the storms as measured by the radar. The higher values for these variables agrees with the hypothesis that storms should be seeded earlier in their evolution rather than later.

The storm property $v166$ (Mean dBz at 6 km: Time to max rate of increase) must be singled out. This property has a large difference between the means of the "seeded" and "unseeded" storms for both models. It is also the only property that has a large difference between the means for both models according to the analyses of variance where all homoscedastic variables were entered into the analyses (21 storm properties in the case of the discriminant analysis and 17 storm properties in the case of the artificial neural network model).

Variable	Discriminant analysis		MLP		True status	
	Unseeded	Seeded	Unseeded	Seeded	Unseeded	Seeded
$v32$: Peak dBz at decision time	46,77	54,36	45,80	53,02	48,70	49,91
$v38$: Echo top: Persistence	-0,03	0,04	-0,04	0,03	0,01	-0,00
$v64$: Storm mass above 6 km: Max rate of increase	0,12	0,31	0,10	0,27	0,13	0,20
$v67$: Storm mass above 6 km: Time to max	4,28	7,77	4,17	6,90	5,36	5,72
$v68$: Storm mass above 6 km: Time to max rate of increase	3,25	5,60	2,75	5,34	4,77	4,22
$v69$: Storm mass above 6 km: Persistence	-0,08	0,22	-0,13	0,17	0,01	0,04
$v99$: Rain flux at 6 km: Time to max	4,71	7,47	4,53	6,85	5,39	5,85
$v118$: Vertical centroid: Time to max	3,16	5,31	2,33	5,36	4,14	4,05
$v120$: Vertical centroid: Persistence	-0,05	0,00	-0,07	0,01	-0,03	-0,03
$v126$: Reflectivity-weighted centroid: Time to max rate of increase	3,10	4,85	2,77	4,63	4,72	3,83
$v127$: Reflectivity-weighted centroid: Persistence	-0,06	0,01	-0,09	0,00	-0,03	-0,03
$v166$: Mean dBz at 6 km: Time to max rate of increase	3,13	6,12	3,06	5,36	5,39	4,37
$v169$: Max height of 45 dBz: Max	6997,90	8745,50	6474,39	8665,48	7827,66	7721,04
$v171$: Max height of 45 dBz: Mean	6047,87	6740,99	5775,42	7472,48	6939,95	6740,99
$v182$: Height of peak dBz: Max ratio	0,079	0,10	0,06	0,13	0,09	0,10

Table 6.13: Storm properties corresponding to both models

Lastly, oneway analyses of variance were performed between the seeded and unseeded storms which were classified as seeded by the discriminant analysis model and the artificial neural network model respectively. Only $v247$, $v252$ and $v262$ - all three summary statistics of the lowest scan as a function of dBz - had large differences between the seeded and unseeded storms for the 35 storm properties identified as having large differences by the discriminant analysis. No differences were found with respect to the 31 storm properties identified by the artificial neural network model, nor for the 15 storm properties which were common to both models. This is as expected, and confirms that the storm properties behaved consistently for the storms which were identified by the two models as seeded, regardless of their true seeding status.

6.7 Conclusion

Both the statistical discriminant analysis model and the artificial neural network model performed well at separating those seeded storms which had apparently reacted positively to seeding and those which had apparently not reacted, although they used different criteria in the process. In view of the relatively small sample size, the artificial neural network model performance was of particular interest.

The analyses of variances, performed with the two model classification of the 58 seeded storms as "seeded" and "unseeded" as factor, identified fifteen storm properties with respect to which large differences were found for both models. Based on these storm properties, indications are that storms appropriate for seeding are still in a growing phase. These properties are readily calculated from the radar measurements in the ten minutes before the seeding decision is taken and may therefore indeed be very useful to identify suitable storms for seeding in future weather modification operations.

Chapter 7

Finale

This chapter provides a summary of the contents of the thesis, and presents proposals for further research initiatives emanating directly from the research conducted, as well as conceived by the stimulation of the research process.

7.1 Summary

7.1.1 Literature study

The extensive literature study on artificial neural network methodology has revealed that the nature of the research in this field and the related applications are strongly interdisciplinary, attracting scientists and technologists from a wide range of disciplines, including computer scientists, mathematicians, meteorologists, electrical and computer engineers, physicists, neuroscientists, psychologists and financial analysts.

Chapters 2 to 4 summarize the results of the literature study in a very specific way. Chapter 2 poses an artificial neural network in the context of its neurological counterpart and compares Statistics with artificial neural networks, while Chapter 3 sets artificial neural networks in a historical context, tracing the evolution of the field over the past few decades. Chapter 4 sets out

and explains the various components of artificial neural network systems in a logical and coherent way. This representation is unique and was extracted and assimilated from the vast body of literature on artificial neural networks, all written from different perspectives by authors involved with artificial neural networks across the spectrum of application disciplines.

7.1.2 Optimization algorithm

The recursive algorithm that was developed to optimize the number of hidden nodes in a feedforward artificial neural network, as explained in Chapter 5, demonstrates how statistical methodology can be applied to develop and refine neural network methodology. This algorithm is an original contribution to the field of artificial neural network methodology that simplifies the process of artificial neural network architecture selection, thereby reducing the number of training sessions necessary to find a model that fits the data adequately.

7.1.3 Modelling

Chapter 6 presents the statistical modelling of weather modification data using both an artificial neural network and a classical statistical technique. The research objective in this chapter was two-pronged. The one goal was to ascertain how well an artificial neural network model performs on a smaller data set in comparison with an analogous statistical technique. The other goal was to address the problem of selecting appropriate storms for seeding in weather modification experiments.

The results of the modelling process indicated that the classification model obtained by the statistical discriminant analysis and the classification model obtained by the multilayer perceptron both performed well at separating those seeded storms which had apparently reacted positively to seeding and those which had apparently not reacted to seeding, although the two techniques used

very different criteria in the process. In view of the relatively small sample size - only 114 storms were included in the experiment - the artificial neural network model performance was of particular interest as most of the practical applications of artificial neural networks typically deal with very large data sets. In this case, in any event, the effectiveness of artificial neural networks as a modelling tool involving smaller data sets has been established.

Furthermore, using the classification results to divide the seeded storms into those storms that resemble unseeded storms and those storms that seemed to have responded positively to seeding, and performing analyses of variance on the storm properties, succeeded in identifying a number of radar-derived variables that may be useful to differentiate between storms suitable for seeding and those that are not. As the calculations are based on the radar measurements taken in the ten minutes before the seeding decision was taken, the variables that have been identified as having possible explanatory power may be calculated by the radar operators in real time before the actual decision to seed a particular storm is taken. This will increase the probability of a positive seeding effect, with obvious monetary benefits.

The work in this chapter, including the formulation of the problem, the approach that has been followed to solve it and the novel modelling application, makes an original contribution to the interdisciplinary fields of Statistics and Artificial Neural Networks as well as to the discipline of Meteorology.

The decision to use an artificial neural network package for the analyses in this chapter, as opposed to programming the artificial neural network algorithm, was motivated by the availability of sophisticated software and the ease of its use. This will inevitably lead to the increased use of these packages by data analysts, in the same way that statistical software is nowadays routinely included in spreadsheets and used extensively by non-statisticians.

7.2 Future research

For a statistician to get involved with artificial neural networks is tantamount to opening a can of worms. Research opportunities abound, both from a theoretical and a practical perspective.

A few proposals, stemming directly and indirectly from the research conducted for this thesis, are briefly outlined below.

- A radial basis function artificial neural network can be used instead of a multilayer perceptron to develop a classification model for storms (cf. Chapter 6). The results of this model can be compared to the results of the discriminant analysis model, as well as to the results of the multilayer perceptron.

- A clustering artificial neural network, e.g. a Kohonen neural network, can be employed to cluster storms that have similar profiles with respect to the 250 radar-derived storm properties (cf. Chapter 6). As in this thesis, the modelling results of the artificial neural network can be compared to the statistical results of a cluster analysis. In order to perform these analyses in SPSS, the data file, which is in a spreadsheet format with cases (i.e. storms) in the rows and variables (i.e. storm properties) in the columns, will have to be transposed

- The modelling results using an artificial neural network software package can be compared to the results of algorithms programmed by the user. This initiative may be marginal to the field of Statistics, but will nonetheless be important in establishing the credentials of artificial neural network software amongst data analysts who rely on these packages.

- The application of artificial neural network models to geostatistical problems is tantalizing as data from this discipline is by its very nature sparse.

Bibliography

[Ackley *et al.*, 1985] Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. 1985. A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, **9**, 147–169.

[Aitkin, 1999] Aitkin, M. 1999. A General Maximum Likelihood Analysis of Variance Components in Generalized Linear Models. *Biometrics*, **55**, 117–128.

[Amari, 1972] Amari, S. I. 1972. Learning Patterns and Pattern Sequences by Self-Organizing Nets of Threshold Elements. *IEEE Transactions on Computers*, **C-21**(11), 1197–1206.

[Amari, 1977] Amari, S. I. 1977. Neural Theory of Association and Concept Formation. *Biological Cybernetics*, **26**, 175–185.

[Amari *et al.*, 1995] Amari, S. I., Murata, N., Müller, K. R., Finke, M., & Yang, H. 1995. *Asymptotic Statistical Theory of Overtraining and Cross-Validation.* Tech. rept. METR95-06. University of Tokyo, Department of Mathematical Engineering and Information, Physics, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan.

[Anderson, 1995] Anderson, J. A. 1995. *An Introduction to Neural Networks.* Cambridge, Massachusetts: MIT Press.

[Anderson *et al.*, 1977] Anderson, J. A., Silverstein, J. W., Rite, S. A., & Jones, R. S. 1977. Distinctive Features, Categorical Perception and Proba-

bility Learning: Some Applications of a Neural Model. *Psychological Review*, **84**, 413–451.

[Anthony & Bartlett, 1999] Anthony, M., & Bartlett, P. L. 1999. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press.

[Barron, 1993] Barron, A. R. 1993. Universal Approximation Bounds for Superpositions of a Sigmoidal Function. *IEEE Transactions on Information Theory*, **39**(3), 930–945.

[Barron, 1994] Barron, A. R. 1994. Approximation and Estimation Bounds for Artificial Neural Networks. *Machine Learning*, 115–133.

[Bates & Watts, 1988] Bates, D. M., & Watts, D. G. 1988. *Nonlinear Regression Analysis and Its Applications.* New York: John Wiley & Sons.

[Bertsekas, 1995a] Bertsekas, D. P. 1995a. *Dynamic Programming and Optimal Control: Volumes 1 and 2.* Belmont, MA: Athena Scientific.

[Bertsekas, 1995b] Bertsekas, D. P. 1995b. *Nonlinear Optimization.* Belmont, Massachusetts: Athena Scientific.

[Bishop, 1995] Bishop, C. M. 1995. *Neural Networks for Pattern Recognition.* New York: Oxford University Press.

[Breslow & Clayton, 1993] Breslow, N. E., & Clayton, D. G. 1993. Approximate Inference in Generalized Linear Mixed Models. *Journal of the American Statistical Association*, **88**(421), 9–25.

[Broomhead & Lowe, 1988] Broomhead, D. S., & Lowe, D. 1988. Multivariable functional interpolation and adaptive networks. *Complex Systems*, **2**, 321–355.

[Burks & Von Neumann, 1947] Burks, A. W., Goldstine H. H., & Von Neumann, J. 1947. *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, Part 1.*

[Cheng & Titterington, 1994] Cheng, B., & Titterington, D. M. 1994. Neural Networks: A Review from a Statistical Perspective. *Statistical Science*, **9**(1), 2–54.

[Cherkassky & Wechsler, 1994] Cherkassky, V., Friedman J. H., & Wechsler, H. 1994. *From Statistics to Neural Networks: Theory and Pattern Recognition Applications.* F: Computer and Systems Sciences. Berlin Heidelberg: Springer-Verlag.

[Cherkassky, 1996] Cherkassky, V. 1996. Comparison of Adaptive Methods for Function Estimation from Samples. *IEEE Transactions on Neural Networks*, **7**(4), 969–984.

[Cohen, 1995] Cohen, P. R. 1995. *Empirical Methods for Artificial Intelligence.* London: MIT Press.

[Cortes & Vapnik, 1995] Cortes, C., & Vapnik, V. N. 1995. Support Vector Networks. *Machine Learning*, **20**, 273–297.

[Cottrell, 1990] Cottrell, G. W. 1990. Extracting Features from Faces using Compression Networks: Face, identity, emotion and gender recognition using holons. *In:* Touretzky, D. (ed), *Connection Models: Proceedings of the 1990 Summer Schools.* San Mateo, CA: Morgan Kaufmann.

[Cover, 1965] Cover, T. M. 1965. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computing*, **EC-14**(14), 326–334.

[Cybenko, 1988] Cybenko, G. 1988. *Continuous valued neural networks with two hidden layers are sufficient.* Tech. rept. Tufts University, Department of Computer Science, Medford, MA.

[Cybenko, 1989] Cybenko, G. 1989. Approximation by Superpositions of a Sigmoidal Function. *Mathematical Control, Signals and Systems*, **2**, 303–314.

[Deco & Obradovic, 1996] Deco, G, & Obradovic, D. 1996. *An Information-Theoretic Approach to Neural Computing.* New York: Springer-Verlag.

[Dixon & Mather, 1986] Dixon, M. J., & Mather, G. K. 1986. *Programme for Atmospheric Water Supply - Phase I, 1983–1986, Volume III.* Tech. rept. 133/3/1986. Water Research Commission, P O Box 824, Pretoria, 0001, South Africa.

[Eberhart & Dobbins, 1990] Eberhart, R. C., & Dobbins, R. W. 1990. Case Study I: Detection of Electroencephalogram Spikes. *In:* Eberhart, R. C., & Dobbins, R. W. (eds), *Neural Networks PC Tools.* San Diego: Academic Press.

[Engelbrecht, 1999] Engelbrecht, A. P. 1999. *Sensitivity Analysis of Multilayer Neural Networks.* PhD dissertation, University of Stellenbosch.

[Engelbrecht & Fletcher, 1999] Engelbrecht, A. P., Cloete I., & Fletcher, L. 1999. Variance Analysis of Sensitivity Information for Pruning Multilayer Feedforward Neural Networks. *In: IEEE IJCNN.* Washinton D.C.: IEEE. Paper 379.

[Fahlman, 1988] Fahlman, S. E. 1988. Faster-Learning Variations on Back-Propagation: An Empirical Study. *Pages 38–51 of:* Touretzky, D., Hinton G., & Sejnowski, T. (eds), *Proceedings of the 1988 Connectionist Models Summer School.*

[Fan & Gijbels, 1995] Fan, J., & Gijbels, I. 1995. *Local polynomial modelling and its applications.* New York: Chapman & Hall.

[Fausett, 1994] Fausett, L. 1994. *Fundamentals of Neural Networks: Architecture, Algorithms and Applications.* New Jersey: Prentice-Hall.

[Fine, 1999] Fine, T. L. 1999. *Feedforward Neural Network Methodology.* New York: Springer-Verlag.

[Fletcher & Engelbrecht, 1998] Fletcher, L., Katkovnik V. Steffens F. E., & Engelbrecht, A. P. 1998. Optimizing the number of hidden nodes of a feedforward artificial neural network. *Pages 1608–1612 of: IEEE World Congress on Computational Intelligence.* New York: IEEE.

[Fletcher & Steffens, 1996] Fletcher, L., & Steffens, F. E. 1996. The Use of Permutation Techniques in Evaluating the Outcome of a Randomized Storm Seeding Experiment. *Journal of Applied Meteorology*, **35**(9), 1546–1550.

[Fletcher & Steffens, 2002a] Fletcher, L., & Steffens, F. E. 2002a. A method of storm selection in weather modification experiments: Part 1. *Pages 59–67 of: Some Statistical Problems in Industry and Science.* Pretoria: Unisa.

[Fletcher & Steffens, 2002b] Fletcher, L., & Steffens, F. E. 2002b. A method of storm selection in weather modification experiments: Part 2. *In: South African Society for Atmospheric Sciences.*

[Fukushima & Miyaka, 1980] Fukushima, K., & Miyaka, S. 1980. Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, **36**(4), 193–202.

[Galkin, 2001] Galkin, I. 2001. *Crash Introduction to Artificial Neural Networks.* Internet: http://ulcar.uml.edu/ iag/CS/Intro-to-ANN.html.

[Geman & Doursat, 1992] Geman, S., Bienenstock E., & Doursat, R. 1992. Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, **4**, 1–58.

[Gill *et al.*, 1981] Gill, P. E., Murray, W., & White, M. H. 1981. *Practical Optimization.* London: Academic Press.

[Green & Carroll, 1978] Green, P. E., & Carroll, J. D. 1978. *Mathematical Tools for Applied Multivariate Analysis.* London: Academic Press, Inc.

[Green & Silverman, 1994] Green, P. J., & Silverman, B. W. 1994. *Nonparametric regression and generalized linear models.* London: Chapman & Hall.

[Grossberg, 1977] Grossberg, S. 1977. Classical and Instrumental Learning by Neural Networks. *Pages 51–141 of: Progress in Theoretical Biology, vol. 3.* New York: Academic Press.

[Hair & Black, 1998] Hair, J. F. Jr., Anderson R. E. Tatham R. L., & Black, W. C. 1998. *Multivariate Data Analysis.* New Jersey: Prentice Hall.

[Hassoun, 1995] Hassoun, M. H. 1995. *Fundamentals of Artificial Neural Networks.* MIT Press.

[Haykin, 1999] Haykin, S. 1999. *Neural Networks: A Comprehensive Foundation.* New Jersey: Prentice Hall.

[Hebb, 1949] Hebb, D. O. 1949. *The Organisation of Behaviour, A Neuropsychological Theory.* New York: John Wiley.

[Hecht-Nielsen, 1988] Hecht-Nielsen, R. 1988. Neuralcomputating: picking the human brain. *IEEE Spectrum*, **25**(3), 36–41.

[Hecht-Nielsen, 1990] Hecht-Nielsen, R. 1990. *Neurocomputing.* Reading: Addison-Wesley.

[Hewitson & Crane, 1994] Hewitson, B. C., & Crane, R. G. 1994. *Neural Nets: Applications in Geography.* Kluwer Academic Publishers.

[Hopfield, 1982] Hopfield, J. J. 1982. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Pages 2554–2558 of: Proceedings of the National Academy of Scientists.*

[Hopfield, 1984] Hopfield, J. J. 1984. Neurons with Graded response have Collective Computational Properties like those of Two State Neurons. *In: Proceedings of the National Academy of Scientists.*

[Hornik & White, 1989] Hornik, K. M., Stinchcombe M., & White, H. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**(5), 359–366.

[Householder & Landahl, 1945] Householder, S. A., & Landahl, H. D. 1945. *Mathematical Biophysics of the Central Nervous System.* Mathematical Biophysics Monograph Series, No. 1. Bloomington, Indiana: Principia Press.

[Hurrich & Tsai, 1989] Hurrich, C. M., & Tsai, C. L. 1989. Regression and time series model selection in small samples. *Biometrika*, 297–307.

[Ismail & Engelbrecht, 2000] Ismail, A., & Engelbrecht, A. 2000. Global optimization algorithms for training product unit neural networks. *Pages 132–137 of: International Joint Conference on Neural Networks IJCNN'2000*, vol. I. IEEE Computer Society, Los Alamitos, CA.

[Jacobs, 1988] Jacobs, R. A. 1988. Increased Rate of Convergence Through Learning rate Adaptation. *Neural Networks*, **5**(4), 295–307.

[Jones *et al.*, 1999] Jones, C., Peterson, P., & Gautier, C. 1999. A New Method for Deriving Ocean Surface Specific Humidity and Air Tempera-

ture: An Artificial Neural Network Approach. *Journal of Applied Meteorology*, **38**(8), 1229–1245.

[Kauermann & Tutz, 1999] Kauermann, R. G., & Tutz, G. 1999. On model diagnostics using varying coefficient models. *Biometrika*, **86**, 119–128.

[Kay & Titterington, 1999] Kay, J. W., & Titterington, D. M. 1999. *Statistics and Neural Networks: Advances at the Interface.* Oxford University Press.

[Kirkpatrick & Vecchi, 1983] Kirkpatrick, S., Gelatt C. D., & Vecchi, M. P. 1983. Optimization by simulated annealing. *Science*, **220**, 671–680.

[Kohonen, 1977] Kohonen, T. 1977. *Associative Memory: A System Theoretic Approach.* Berlin: Springer-Verlag.

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

[Kohonen, 1988] Kohonen, T. 1988. The "Neural" Phonetic Typewriter. *Computer*, **21**(3), 11–22.

[Kohonen, 1990] Kohonen, T. 1990. The Self-organizing Map. *Proceedings of the IEEE*, **78**(9), 1464–1480.

[Landahl, 1961] Landahl, H. D. 1961. A note on mathematical models for the interaction of neural elements. *Bulletin of Mathematical Biophysics*, **23**, 91–97.

[Landahl & Pitts, 1943] Landahl, H. D., McCulloch W. S., & Pitts, W. 1943. A statistical consequence of the logical calculus of nervous nets. *Bulletin of Mathematical Biophysics*, **5**, 135–137.

[Lang & Hinton, 1990] Lang, K. J., Waibel A. H., & Hinton, G. E. 1990. A Time-delay Neural Network Architecture for Isolated Word Recognition. *Neural Networks*, **3**, 33–43.

[LeCun & Solla, 1990] LeCun, Y., Denker J. S., & Solla, S. A. 1990. Optimal Brain Damage. *Pages 598–605 of:* Touretzky, D. S. (ed), *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, vol. 2. San Mateo, California: Morgan Kaufmann.

[LeCun *et al.*, 1989] LeCun, Y., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, **1**(4).

[Lettvin & Pitts, 1959] Lettvin, J. Y., Maturana H. R. McCulloch W. S., & Pitts, W. 1959. What the Frog's Eye Tells the Frog's Brain. *Proceedings of the IRE*, **47**(11), 1940–1951.

[Liu *et al.*, 2001] Liu, H., Chandrasekar, V., & Xu, G. 2001. An Adaptive Neural Network Scheme for Radar Rainfall Estimation from WSR-88D Observations. *Journal of Applied Meteorology*, **40**(11), 2038–2050.

[Martin, 1995] Martin, E. B. 1995. *Neural Networks for Chemical Engineers.* Elsevier Science B.V.

[Mather & Fletcher, 1997] Mather, G. K., Terblanche D. E. Steffens F. E., & Fletcher, L. 1997. Results of the South African Cloud-seeding Experiment using Hygroscopic Flares. *Journal of Applied Meteorology*, **36**(11), 1433–1447.

[McClelland & Rumelhart, 1986] McClelland, J. L., & Rumelhart, D. E. 1986. *Parallel Distributed Processing : Explorations in the Microstructure of Cognition : Foundations.* Cambridge: MIT Press and PDP Research Group.

[McClelland & Rumelhart, 1988] McClelland, J. L., & Rumelhart, D. E. 1988. *Explorations in Parallel Distributed Processing.* Cambridge: MIT Press.

[McCulloch, 1962] McCulloch, W. S. 1962. Symbolic representation of the neuron as an unreliable function. *Pages 91–94 of: Principles of Self-Organization (Trans. Sympos. Univ. Illinois, 1961).* New York: Pergamon.

[McCulloch, 1965] McCulloch, W. S. 1965. *Embodiments of mind.* Cambridge, Massachusetts: M.I.T. Press.

[McCulloch & Pitts, 1943] McCulloch, W. S., & Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5**(6,12a), 115–133.

[Minsky, 1954] Minsky, M. L. 1954. *Neural Nets and the Brain Model Problem.* PhD Dissertation in Mathematics, Princeton University.

[Minsky & Papert, 1988 c.1969] Minsky, M. L., & Papert, S. A. 1988 c.1969. *Perceptrons : An Introduction to Computational Geometry.* MIT Press.

[Mira & Sandoval, 1995] Mira, J., & Sandoval, F. 1995. *From Natural to Artificial Neural Computation.* Lecture Notes in Computer Science 930. Springer. International Workshop on Artificial Networks, Malaga-Torremolinos, Spain.

[Mitchell, 1997] Mitchell, T. M. 1997. *Machine Learning.* McGraw-Hill.

[Narasimhan *et al.*, 2000] Narasimhan, R., Keller, J., Subramaniam, G., & Potter, W. P. 2000. Ozone Modeling Using Neural Networks. *Journal of Applied Meteorology*, **39**(3), 291–296.

[Neal, 1992] Neal, R. M. 1992. Connectionist Learning of Belief Networks. *Artificial Intelligence*, **56**, 71–113.

[Nguyen & Widrow, 1990] Nguyen, D., & Widrow, B. 1990. Improving the Learning Speed of Two-Layer Neural Networks by Choosing Initial Values

of the Adaptive Weights. *Pages 21–26 of: Proceedings of the International Joint Conference on Neural Networks, III.* San Diego: IEEE Press.

[Nillson, 1990] Nillson, N. J. 1990. *The Mathematical Foundations of Learning Machines.* San Mateo, California: Morgan-Kaufmann Publishers. Originally published as Learning Machines: Foundations of Trainable Pattern Classifiers, McGraw-Hill, New York, 1965.

[Pitts, 1942] Pitts, W. 1942. The linear theory of neuron networks: the static problem. *Bulletin of Mathematical Biophysics*, **4**, 169–175.

[Pitts, 1943] Pitts, W. 1943. The linear theory of neuron networks: the dynamic problem. *Bulletin of Mathematical Biophysics*, **5**, 23–31.

[Pitts & McCulloch, 1947] Pitts, W., & McCulloch, W. S. 1947. How We Know Universals: The Perception of Auditory and Visual Forms. *Bulletin of Mathematical Biophysics*, **9**, 127–147.

[Rashevsky, 1938] Rashevsky, N. 1938. *Mathematical Biophysics.* Chicago: University of Chicago Press.

[Ratkowsky, 1983] Ratkowsky, D. A. 1983. *Nonlinear Regression Modeling.* New York: Marcel Dekker, Inc.

[Refenes & Francis, 1994] Refenes, A. N., Zapranis A., & Francis, G. 1994. Stock Performance Modeling Using Neural Networks: A Comparative Study With Regression Models. *Neural Networks*, **7**(2), 375–388.

[Rosenblatt, 1958] Rosenblatt, F. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, **65**(6), 386–408.

[Rosenblatt, 1962] Rosenblatt, F. 1962. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Washington, D.C.: Spartan Books.

[Samuel, 1959] Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**(3), 210–223.

[Samuel, 1967] Samuel, A. L. 1967. Some studies in machine learning using the game of checkers, Part II. *IBM Journal of Research and Development*, **1**(4), 601–618.

[Saul & Jordan, 1996] Saul, L. K., Jakkolla T., & Jordan, M. I. 1996. Mean-field Theory for Sigmoid Belief Networks. *Journal of Artificial Intelligence Research*, **4**, 61–76.

[Shimbel & Rapoport, 1948] Shimbel, A., & Rapoport, A. 1948. A Statistical Approach to the Theory of the Central Nervous System. *Bulletin of Mathematical Biophysics*, **10**, 41–55.

[Silverman & Dracup, 2000] Silverman, D., & Dracup, J. A. 2000. Artificial Neural Networks and Long-Range Precipitation Prediction in California. *Journal of Applied Meteorology*, **39**(1), 57–66.

[Smith, 1993] Smith, M. 1993. *Neural Networks for Statistical Modelling.* New York: Van Nostrand Reinhold.

[Smolensky & Rumelhart, 1996] Smolensky, P., Mozer M. C., & Rumelhart, D. E. 1996. *Mathematical Perspectives on Neural Networks.* New Jersey: Lawrence Erlbaum Associates, Inc.

[Snyman, 1983] Snyman, J. A. 1983. An Improved Version of the Original LeapFrog Dynamic Method for Unconstrained Minimization: LFOP1(b). *Applied Mathematical Modelling*, **7**, 216–218.

[SPSS, 1995] SPSS. 1995. *Neural Connection 1.0 User's Guide.* SPSS Inc./Recognition Systems Inc.

[Steffens, 1999] Steffens, F. E. 1999. Using discriminant analysis to distinguish between storms which reacted positively to seeding and storms which did not react positively. *Pages 41–44 of: Seventh WMO Scientific Conference on Weather Modification.* WMO/TD no. 936, no. 31. World Meteorological Organization.

[Steffens & Fletcher, 2002] Steffens, F. E., & Fletcher, L. 2002. Exploring Information in a Weather Modification Data Set to Identify and Characterize Storms that Reacted Positively to Seeding. *In: C. Warren Neel Conference on The New Frontiers of Statistical Data Mining, Knowledge Discovery, and E-Business Intelligence.* Chapman and Hall/CRC.

[Steppe & Rogers, 1996] Steppe, J. M., Bauer K. W., & Rogers, S. K. 1996. Integrated Feature and Architecture Selection. *IEEE Transactions on Neural Networks*, **7**(4), 1007–1014.

[Stinchcombe & White, 1989] Stinchcombe, M., & White, H. 1989. Universal approximation using feedforward networks, with non-sigmoid hidden layer activation functions. *Pages 613–617 of: Proceedings of the 1989 International Joint Conference on Neural Networks.* Washington DC: IEEE Press.

[Turing, 1936–37] Turing, A. M. 1936–37. On Computable Numbers, with an Application to the Entscheidungsproblem. *Pages 230–265 of: Proceedings of the London Mathematical Society.* 2.

[Uttley, 1956] Uttley, A. M. 1956. Conditional Probability Machines and Conditional Reflexes. *Pages 253–285 of:* Shannon, C. E., & McCarthy, J. (eds), *Automata Studies.* Princeton: Princeton University Press.

[Uttley, 1959a] Uttley, A. M. 1959a. Conditional Probability Computing in a Nervous System. *In: Mechanization of Thought Processes.* London: H. M. Stationery Office.

[Uttley, 1959b] Uttley, A. M. 1959b. The Design of Conditional Probability Computers. *Information and Control*, **2**(1), 1–24.

[Van den Bergh & Engelbrecht, 2000] Van den Bergh, F., & Engelbrecht, A. 2000. Cooperative Learning in Neural Networks using Particle Swarm Optimizers. *South African Computer Journal*, **26**, 84–90.

[Vapnik, 1995] Vapnik, V. N. 1995. *The Nature of Statistical Learning Theory.* New York: Springer-Verlag.

[Venkatasubramanian & Rengaswamy, 1995] Venkatasubramanian, V., & Rengaswamy, R. 1995. *Neural Networks for Chemical Engineers.* Elsevier Science B.V.

[Vidyasagar, 1997] Vidyasagar, M. 1997. *A Theory of Learning and Generalization: With Applications to Neural Networks and Control Systems.* London: Springer.

[Von der Malsburg, 1973] Von der Malsburg, C. 1973. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, **14**, 85–100.

[Von Neumann, 1958] Von Neumann, J. 1958. *The Computer and the Brain.* New Haven: Yale University Press.

[Wei, 1992] Wei, C. Z. 1992. On predictive least squares principles. *The Annals of Statistics*, **20**(1), 1–42.

[Werbos, 1974] Werbos, P. 1974. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University.

[White, 1989] White, H. 1989. Neural-Network Learning and Statistics. *AI Expert*, **4**(Dec.), 48–52.

[White, 1992] White, H. 1992. *Artificial Neural Networks.* Cambridge: Blackwell.

[Widrow, 1962] Widrow, B. 1962. Generalization and Information Storage in Networks of Adaline Neurons. *Pages 435–461 of:* M. C. Jovitz, G. T. Jacobi, & Goldstein, G. (eds), *Self-organizing Systems.* Washington D.C.: Spartan Books.

[Widrow & Hoff, 1960] Widrow, B., & Hoff, M. E. 1960. Adaptive Switching Circuits. *Pages 96–104 of: 1960 IRE Western Electric Show and Convention (WESCON) Record.* New York: IRE. Reprinted in Anderson & Rosenfeld [1988], pp. 126-134.

[Willshaw & Von der Malsburg, 1976] Willshaw, D. J, & Von der Malsburg, C. 1976. How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London Series B*, **194**, 431–445.

[Xue & Tompkins, 1990] Xue, Q, Hu Y., & Tompkins, W. J. 1990. Analyses of the Hidden Units of Back-Propagation Model by Singular Value Decomposition (SVD). *IJCNN Conference*, I–739–I–742.

[Zurada, 1992] Zurada, J. C. 1992. *Introduction to Artificial Neural Systems.* West Publishing Company.

Appendix A

Description of the 250 NPRP storm properties

A. Seven variables relating to the identification of the storm:

1. Year
2. Month
3. Day
4. Sequence (21 = Nelspuit; 33 = Bethlehem)
5. Track number
6. Envelop number (for the randomization of Seed/No seed)
7. Seed (0 = Not seeded; 1 = Seeded)

B. Fourteen measures of rain mass:

1 - 7: RAIN MASS AT THE LOWEST SCAN

1. 0 - 10 minutes period before seeding decision
2. 0 - 10 minutes after seeding decision (seeding taking place in this period)
3. 10-20 minutes after the seeding decision (seeding may still be taking place)
4. 20-30 minutes after the seeding decision
5. 30-40 minutes after the seeding decision

6. 40-50 minutes after the seeding decision
7. 50-60 minutes after the seeding decision

8 - 14: RAIN MASS AT 6KM ALTITUDE

8. 0 - 10 minutes period before seeding decision
9. 0 - 10 minutes after seeding decision (seeding taking place in this period)
10. 10-20 minutes after the seeding decision (seeding may still be taking place)
11. 20-30 minutes after the seeding decision
12. 30-40 minutes after the seeding decision
13. 40-50 minutes after the seeding decision
14. 50-60 minutes after the seeding decision

C: 250 measurements 10 minutes before the seeding decision

1. Duration of period (10 minutes) (in decimal hours)
2. Time since storm origin (decimal hours)
3. Storm speed in km/hr
4. Direction of movement (degrees)
5. Mean X co-ordinate (measured from the radar)
6. Mean Y co-ordinate (measured from the radar)
7. Mean range (distance from the radar)
8. Decision time
9. Volume at decision time
10. Peak dBz at decision time

ECHO TOPS (11-17)

11. Max
12. Max rate of increase
13. Mean
14. Time to max

15. Time to max rate of increase
16. Persistence (= max value in the period / max value at beginning or end of the period)
17. Max ratio (= max value in the period / value at beginning)

STORM DEPTH (18-24)

18. Max
19. Max rate of increase
20. Mean
21. Time to max
22. Time to max rate of increase
23. Persistence (= max value in the period / max value at beginning or end of the period)
24. Max ratio (= max value in the period / value at beginning)

STORM VOLUME (25-32)

25. Max
26. Max rate of increase
27. Time integral
28. Mean
29. Time to max
30. Time to max rate of increase
31. Persistence (= max value in the period / max value at beginning or end of the period)
32. Max ratio (= max value in the period / value at beginning)

TOTAL STORM MASS (33-40)

33. Max
34. Max rate of increase
35. Time integral
36. Mean
37. Time to max

38. Time to max rate of increase
39. Persistence (= max value in the period / max value at beginning or end of the period)
40. Max ratio (= max value in the period / value at beginning)

MASS OF THE STORM ABOVE 6 KM (41-48)

41. Max
42. Max rate of increase
43. Time integral
44. Mean
45. Time to max
46. Time to max rate of increase
47. Persistence (= max value in the period / max value at beginning or end of the period)
48. Max ratio (= max value in the period / value at beginning)

STORM AREA AT LOWEST SCAN (3 DEGREES) (49-56)

49. Max
50. Max rate of increase
51. Time integral
52. Mean
53. Time to max
54. Time to max rate of increase
55. Persistence (= max value in the period / max value at beginning or end of the period)
56. Max ratio (= max value in the period / value at beginning)

STORM AREA AT 6 KM (57-64)

57. Max
58. Max rate of increase
59. Time integral
60. Mean

61. Time to max

62. Time to max rate of increase

63. Persistence (= max value in the period / max value at beginning or end of the period)

64. Max ratio (= max value in the period / value at beginning)

RAIN FLUX AT LOWEST SCAN (3 DEGREES) (65-72)

65. Max

66. Max rate of increase

67. Time integral [=Rain mass at lowest scan]

68. Mean

69. Time to max

70. Time to max rate of increase

71. Persistence (= max value in the period / max value at beginning or end of the period)

72. Max ratio (= max value in the period / value at beginning)

RAIN FLUX AT 6 KM (73-80)

73. Max

74. Max rate of increase

75. Time integral [=Rain mass at 6 km]

76. Mean

77. Time to max

78. Time to max rate of increase

79. Persistence (= max value in the period / max value at beginning or end of the period)

80. Max ratio (= max value in the period / value at beginning)

PRECIPITABLE WATER (81-88)

81. Max

82. Max rate of increase

83. Time integral

84. Mean
85. Time to max
86. Time to max rate of increase
87. Persistence (= max value in the period / max value at beginning or end of the period)
88. Max ratio (= max value in the period / value at beginning)

RATIOS (89-92)

89. PROP(89)=PROP(35)/PROP(67)
90. PROP(90)=PROP(43)/PROP(75)
91. PROP(91)=PROP(36)/PROP(67)
92. PROP(92)=PROP(44)/PROP(74)

VERTICAL CENTROID(93-99)

93. Max
94. Max rate of increase
95. Mean
96. Time to max
97. Time to max rate of increase
98. Persistence (= max value in the period / max value at beginning or end of the period)
99. Max ratio (= max value in the period / value at beginning)

REFLECTIVITY-WEIGHTED CENTROID (100-106)

100. Max
101. Max rate of increase
102. Mean
103. Time to max
104. Time to max rate of increase
105. Persistence (= max value in the period / max value at beginning or end of the period)

106. Max ratio (= max value in the period / value at beginning)

REFLECTIVITY-WEIGHTED CENTROID MINUS CENTROID (107-113)

107. Max

108. Max rate of increase

109. Mean

110. Time to max

111. Time to max rate of increase

112. Persistence (= max value in the period / max value at beginning or end of the period)

113. Max ratio (= max value in the period / value at beginning)

PROPERTIES(114-115)

114. PROP(114)=Discriminant function (of unknown value)

115. PROP(115)=PROP(67)/PROP(51)

RAINFLUX AT TIMES 0 AND 10 (116-118)

116. Max

117. Max rate of increase

118. Max ratio

PEAK dBz OVER WHOLE VOLUME (119-125)

119. Max

120. Max rate of increase

121. Mean

122. Time to max

123. Time to max rate of increase

124. Persistence (= max value in the period / max value at beginning or end of the period)

125. Max ratio (= max value in the period / value at beginning)

MEAN dBz OVER WHOLE VOLUME (126-132)

126. Max

127. Max rate of increase
128. Mean
129. Time to max
130. Time to max rate of increase
131. Persistence (= max value in the period / max value at beginning or end of the period)
132. Max ratio (= max value in the period / value at beginning)

MEAN dBz AT LOWEST SCAN (3 DEG.) (133-139)

133. Max
134. Max rate of increase
135. Mean
136. Time to max
137. Time to max rate of increase
138. Persistence (= max value in the period / max value at beginning or end of the period)
139. Max ratio (= max value in the period / value at beginning)

MEAN dBz AT 6 KM (140-146)

140. Max
141. Max rate of increase
142. Mean
143. Time to max
144. Time to max rate of increase
145. Persistence (= max value in the period / max value at beginning or end of the period)
146. Max ratio (= max value in the period / value at beginning)

MAX HEIGHT OF 45 dBz (147-153)

147. Max
148. Max rate of increase
149. Mean

150. Time to max

151. Time to max rate of increase

152. Persistence (= max value in the period / max value at beginning or end of the period)

153. Max ratio (= max value in the period / value at beginning)

HEIGHT OF PEAK dBz (154-160)

154. Max

155. Max rate of increase

156. Mean

157. Time to max

158. Time to max rate of increase

159. Persistence (= max value in the period / max value at beginning or end of the period)

160. Max ratio (= max value in the period / value at beginning)

THE FOLLOWING STATISTICS ARE DESIGNED TO CHARACTERIZE THE STORM USING THE HEIGHT DISTRIBUTION OF MASS AND DBZ, AND THE DISTRIBUTION OF 3 DEG AREA AND VOLUME WITH DBZ.

THE PROGRAM COMPUTES THE MASS AND PEAK DBZ AT VARIOUS HEIGHTS, AND THE DISTRIBUTION OF 3 DEG AREA AND VOLUME WITH DBZ. IN ORDER TO SUMMARIZE THESE DISTRIBUTIONS. THE FOLLOWING ARE COMPUTED FOR EACH:

- MEAN
- STANDARD DEVIATION
- NEGATIVE OF SKEWNESS
- MODE

FOR EACH STATISTIC THE FOLLOWING ARE COMPUTED:

- MEAN
- MAX
- MIN
- MAX RATE OF INCREASE
- MAX RATE OF DECREASE

MASS AS A FUNCTION OF HEIGHT (161-180)

DBZ AS A FUNCTION OF HEIGHT (181-200)

VOLUME AS A FUNCTION OF DBZ (201-220)

3 DEG AREA AS A FUNCTION OF DBZ (221-240)

DAY PROPERTIES (241-250)

The next 10 are **properties of the day** rather than of the storm:

241. Average mixing ratio below 60 MB
242. Temp. CCL
243. DT 500 MB
244. Temp. ratio
245. Number of tracks for day
246. Cum. ATI for day
247. Max. vol. in day
248. Max. ROI of vol. in day
249. Max. top in day
250. Max. dBz in day

Appendix B

Missing value analysis

B.1 Table of estimated missing values

Storm	Missing values	Substitute	
		rml	*rmc*
16	*rml*7; *rmc*7	68	0
48	*rml*6; *rmc*6	80	40
	*rml*7; *rmc*7	100	0
59	*rml*7; *rmc*7	230	200
100	*rml*7; *rmc*7	380	175
108	*rml*6; *rmc*6	0	0
	*rml*7; *rmc*7	0	0

Table B.1: Additional storms with missing values

B.2 Graphs used to estimate missing values

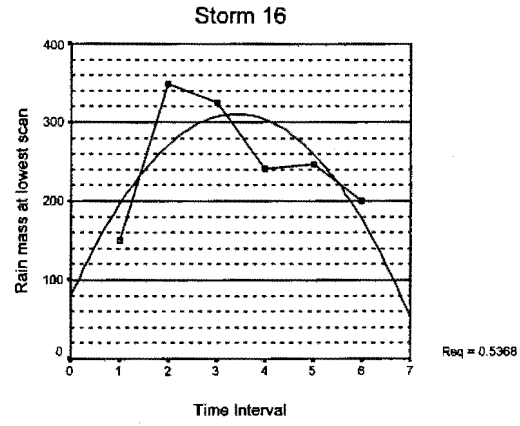

Figure B-1: Rain mass at lowest scan for storm 16

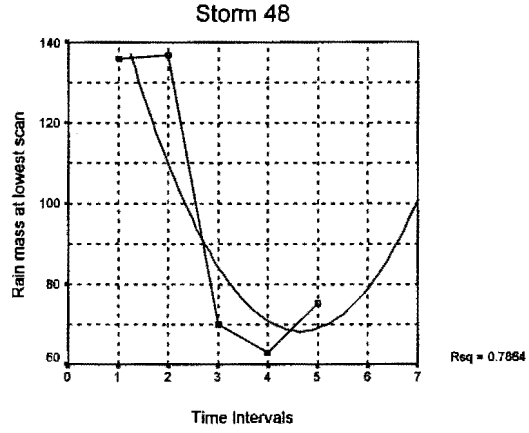

Figure B-2: Rain mass at lowest scan for storm 48

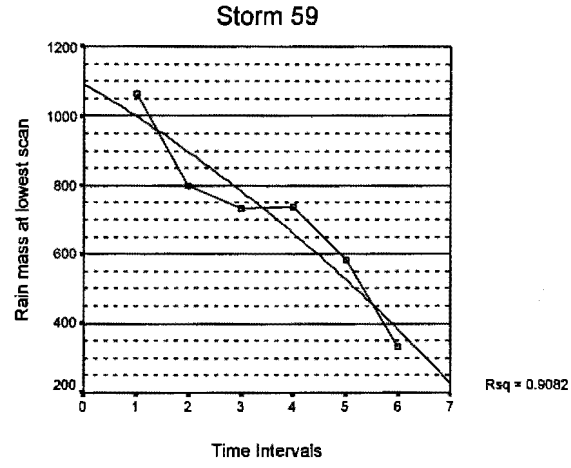

Figure B-3: Rain mass at lowest scan for storm 59

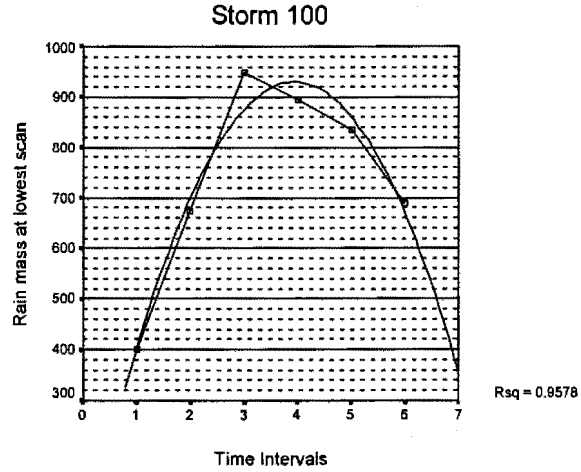

Figure B-4: Rain mass at lowest scan for storm 100

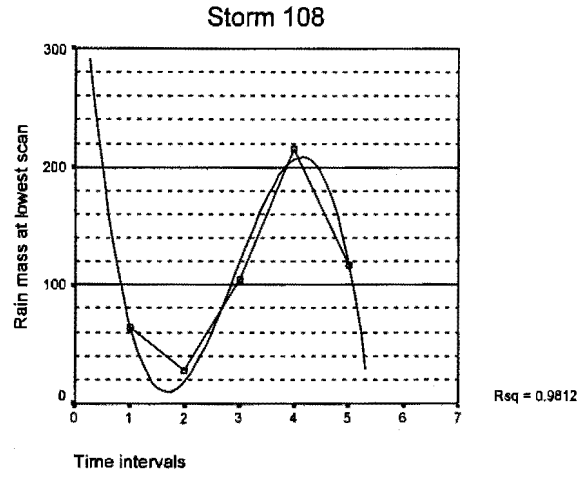

Figure B-5: Rain mass at lowest scan for storm 108

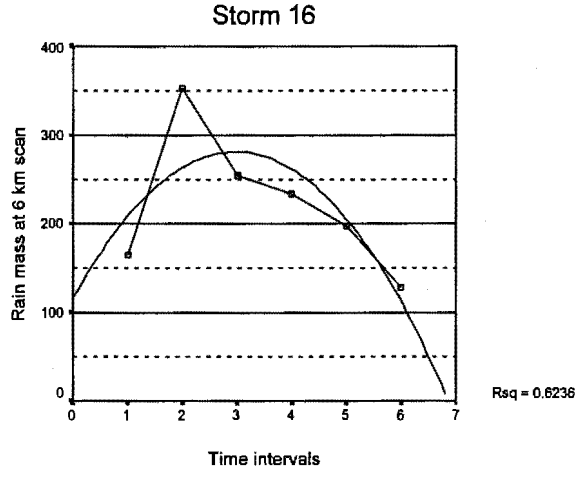

Figure B-6: Rain mass at 6 km scan for storm 16

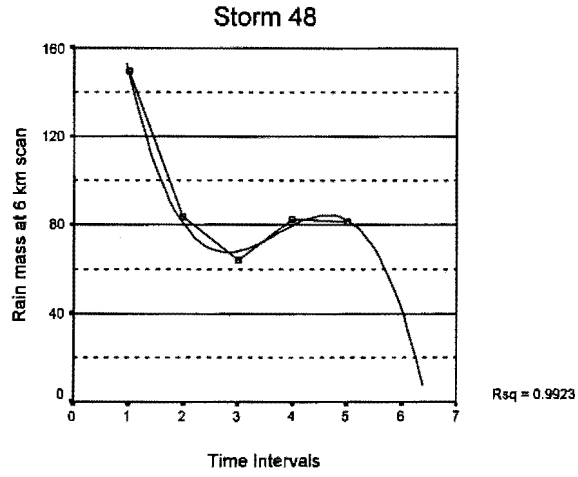

Figure B-7: Rain mass at 6 km scan for storm 48

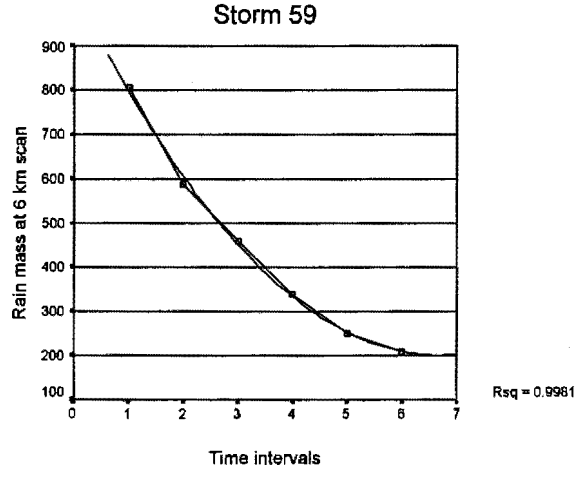

Figure B-8: Rain mass at 6 km scan for storm 59

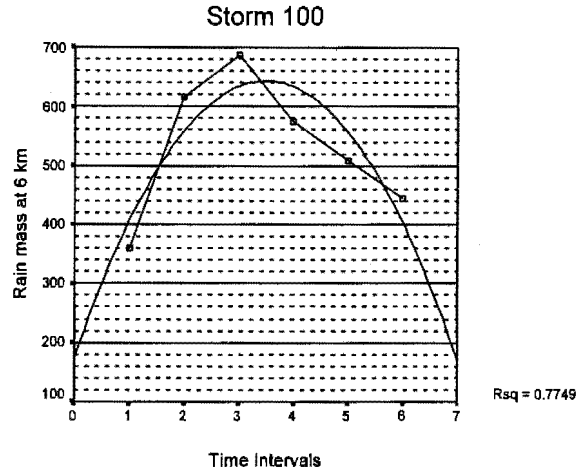

Figure B-9: Rain mass at 6 km for storm 100

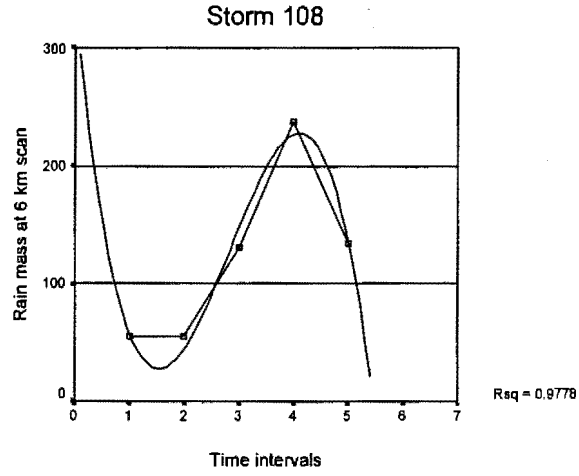

Figure B-10: Rain mass at 6 km for storm 108