Minimum Description Length, Regularisation and Multi-Modal Data

JOHN CONRAD VAN DER REST

Doctor Of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

November 1995

THE UNIVERSITY OF ASTON IN BIRMINGHAM

Minimum Description Length, Regularisation and Multi-Modal Data

JOHN CONRAD VAN DER REST

Doctor Of Philosophy, 1995

Thesis Summary

Conventional feed forward Neural Networks have used the sum-of-squares cost function for training. A new cost function is presented here with a description length interpretation based on Rissanen's Minimum Description Length principle. It is a heuristic that has a rough interpretation as the number of data points fit by the model. Not concerned with finding optimal descriptions, the cost function prefers to form minimum descriptions in a naive way for computational convenience. The cost function is called the *Naive Description Length* cost function. Finding minimum description models will be shown to be closely related to the identification of clusters in the data. As a consequence the minimum of this cost function approximates the most probable mode of the data rather than the sum-of-squares cost function that approximates the mean. The new cost function is shown to provide information about the structure of the data. This is done by inspecting the dependence of the error to the amount of regularisation. This structure provides a method of selecting regularisation parameters as an alternative or supplement to Bayesian methods. The new cost function is tested on a number of multi-valued problems such as a simple inverse kinematics problem. It is also tested on a number of classification and regression problems. The mode-seeking property of this cost function is shown to improve prediction in time series problems. Description length principles are used in a similar fashion to derive a regulariser to control network complexity.

Keywords: Generalisation, Minimum Description Length, Regularisation, Time Series Prediction.

To my Mother and Father

Acknowledgements

I would like to express my sincere thanks to Richard Rohwer my supervisor for his constant help and encouragement throughout my graduate studies at Aston. As well as a great source of knowledge in Neural Networks he has always made himself available and willing to stop whatever he was doing to discuss my work.

None of this work would have been possible without money and resources and I would like to thank David Bounds for giving me the opportunity to do research at Aston and to be part of an expanding group in the field of Neural Networks. I would also like to thank Chris Bishop and David Lowe for spending money on computers and making them accessible to the graduate students and putting up with some of the large programs that I have run on them. I thank them and Ian Nabney for organising and arranging the many seminars that have been interesting and helpful in my research. I would also like to thank the Department of Computer Science for giving me money to attend the IEE conference in Artificial Neural Networks at Cambridge in 1995.

Without access to a good computer system productivity would have given way to frustration and I would like to thank Phil Barrett for providing part-time support along with his Ph.D. studies. I also thank Richard Lister for his dedicated support over the last year. A relaxed and good-humoured office is essential to any working environment and for this I thank my fellow Ph.D. students Jason Price, Mike Tipping, Cazhaow Qazaz, Conor Doherty and Markus Svenson. They have always been willing to stop and help answer so many of those varied questions that arise from day to day.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the 14th century the English Philosopher William of Ockham stated his well-known law of economy. He employed it frequently and sharply against the Scholastic philosophers of his day. His statement that "plurality should not be assumed without necessity" became known as Ockham's razor. People like Thomas Bayes and Claude Shannon later provided a theoretical justification and implementation of this principle. Bayes' Theorem provided a statistical technique for calculating the probability of the validity of a proposition given a prior estimate of its probability and new relevant data. Shannon's Information Theory quantified the amount of information required to convey such propositions. The ideas of these three men have had a profound influence on the rapidly growing field of Neural Computation. They have given increasing importance to selecting models of data that have a Minimum Description Length (MDL) where data can be reconstructed with the minimum of cost. This thesis continues the work into finding Neural Network models that are optimised for simplicity using the principles and theories of Ockham, Bayes and Shannon. Most of the thesis concentrates on finding simple descriptions of the unexplained data.

These principles are used to develop a new cost function for training Neural Networks. The cost function has a description length interpretation. It is not concerned with finding optimal minimum description models of data, preferring to use the principles just described to form minimum descriptions in a naive way for computational convenience. The cost function is called the Naive Description Length (NDL) cost function.

It will be shown that finding minimum description models is closely related to the identification of clusters in the data. As a consequence of this the cost function will be shown to have a mode-seeking property instead of the mean-seeking property of the sum-of-squares cost function used in conventional Neural Network training. This makes it particularly useful in solving interpolation problems that are multi-modal. Through a systematic method of regularisation the underlying structure in the data can be identified. This is shown for regression as well as classification problems. The ability of the cost function to find the most probable mode in data mappings allows an interesting treatment of time series problems. Description lengths are used to describe the errors that a network makes. They are also used to describe the network architecture. The NDL weights regulariser is developed to minimise network complexity.

1.1 Plan of the Thesis

Review

This thesis begins with a review of work related to the new cost function developed here. The review describes the background for this work and explains the basic ideas and issues involved. A knowledge of the basic parts that make up a Neural Network is assumed. Only a brief summary of their main features and advantages is given. This thesis presents a new cost function with a description length interpretation. The different types of representation that a Neural Network can form are described. The idea of finding a set of features in the data with which to reconstruct the data is explained. The representational system defined by a Neural Network must be accurate and efficient. This leads to the explanation of Information Theory to quantify the succinctness of a Neural Network representation. A different interpretation of data model complexity is given with the amount of information used to code the data. The idea of the 'best' model that fits the data and Ockham's razor is discussed. Conventional feed forward networks have used the sum-of-squares error as the cost function in back-propagation networks. The basic features of the sum-of-squares cost function as an error correction form of learning are described, concentrating on its ability to fit the mean of a set of data. The learning process is described and how it must cope with data that is 'ill-posed'. To solve such problems a network must be able to generalise well. The mean-approximation property of the sum-of-squares cost function has been found useful in many applications. The generalisation ability of a network and regularisation techniques are described. The new cost function presented here also has the property of being able to approximate the most probable mode in a data set, leading to quite different generalisations of the data. Other methods are described that improve generalisation through pruning, constructive algorithms, feature extraction and dimensionality reduction. The approach of finding the 'best' model is explained from the Bayesian view of regularisation parameter selection. It will be shown in later chapters that this cost function will aid in the process of achieving the right amount of regularisation controlling the complexity of a network. This is so that optimum generalisation can be achieved for unseen data.

The Naive Description Length cost function

The issues involved in generalisation discussed in the review are described further in the context of multi-modal data. A further interpretation of the new cost function is given, that it extracts the deterministic part of some data containing (in a reasonable sense) a deterministic part and a random part. The NDL cost function is then discussed in detail. A simple multi-modal problem is used to describe the basic features and uses of the NDL cost function. The NDL cost function can be interpreted as the number of data points that fit the model well. It has the property of being a non-mean-approximating cost function with minimum values at the modes of a set of data. It is used on an unsupervised learning problem where the minimum description length feature of the cost function is shown to identify clusters in the data. Supervised learning is viewed as learning a number of unsupervised learning tasks. The cost function is then used in a real-life example of a problem containing multi-modal data, that being the Inverse Kinematics problem. Here it is shown that a non-mean approximating cost function is needed instead of a mean-approximating cost function. The cost function is used together with regularisation to indicate different levels of structure in some data. The dependence of the error on the value of the regularisation parameter is examined in what is called an α-survey.

Regression problems

Regularisation surveys using the NDL cost function can identify levels of structure in the data. These surveys can be used as an aid to choose the value of a regularisation parameter that will give good performance for a test set. The generation of regularisation surveys is explained. Regularisation surveys are used to choose the regularisation parameters for a simple noisy parabola problem. These surveys are compared to similar surveys using the conventional sum-of-squares cost function. Data sets with differing structure in the data are generated to show potential problems with the NDL cost function. Real-life data sets are then used to test the NDL cost function.

Classification problems

The NDL cost function is applied to classification problems with the aim of choosing a good value for the regularising parameter. Regularisation surveys using the NDL cost function are compared to similar surveys using the Cross-Entropy cost function. The reasons for expecting good results for classification problems are discussed and a number of real-life problems are examined.

Time Series prediction

The mode-seeking feature of the NDL cost function is applied to time series prediction problems. A brief description of how the time series experiments were conducted is given. The NDL cost function is used to make predictions on a multi-modal time series. The features of the predictions and the nature of the errors are discussed. A new method of assessing the confidence of predictions naturally arises from the properties of the NDL cost function. The generalisation capability and the number of time-steps used in training is discussed. This chapter ends with experiments carried out on some real-life data sets.

Adaptive NDL

The potential problems with the NDL cost function in training are discussed. Some artificial data sets are described which cause problems in NDL cost function training. An Adaptive NDL cost function is described in detail. How the Adaptive NDL cost function would give better results for these artificial data sets is explained.

NDL Weights

The NDL cost function is used to provide a minimum description length interpretation of the data. A description length interpretation is made for the weights in this chapter. A regulariser is derived in a similar way to the cost function used to model the data. It will be shown that the NDL weights regulariser is a more principled version of the weight elimination method of Weigend *et al.* [89]. It is shown that the NDL weights regulariser has similar properties to weight elimination. The NDL weights regulariser is compared to the weight-decay method of network complexity control. Its properties are shown in regularisation surveys of a simple problem. Finally the NDL cost functions for the data and the NDL weights regulariser are used together.

Conclusions

All the results are summarised and conclusions are given. Some issues not addressed in this thesis are explained. These conclusions are followed by possible future work that could be done using the NDL cost function for modelling the data and controlling network complexity.

1.2 Notation

Several terms and parameters are used in this thesis. Their symbols and descriptions are listed below. The NDL cost function is used in two ways. It is used in describing a description length interpretation of the errors of the output from the network to the targets. It is also used in describing a description length interpretation for the network weights.

Symbol	**Meaning**
x_i	An individual input to a network.
y_i	An individual output from a network.
t_i	An individual target.
$\mathbf{x}$	An input vector.
$\mathbf{y}$	An output vector.
$\mathbf{t}$	A target vector.
X	A set of input vectors.
Y	A set of output vectors.
T	A set of target vectors.
NDL	Naive Description Length.
ϵ	The accuracy within which errors have significance.
Weight ϵ	The accuracy within which a weight is close to zero.
NDL data error	The error from the cost function when it is used to model the data.
NDL network error	The error from the cost function when it is used as a regulariser.
α-survey	A graph showing the dependence of the error on the amount of regularisation in a network.
Training α-survey	A graph showing the error against regularisation in training.
Test α-survey	A graph showing the error against regularisation in testing.
MLP	Multi-Layer-Perceptron.

Chapter 2

Review

Research into Neural Networks has seen a remarkable growth over the last ten years. This is because of their many features that have proved beneficial over traditional methods developed in the science of Artificial Intelligence (AI). The primary feature of artificial Neural Networks is their ability to learn from and adapt to their environment. Instead of knowledge being explicitly stated as a set of rules as the symbolic AI approach adopts, the knowledge is implicitly stored within the weight connections of a network. Thus a Neural Network is a parallel distributed processor of information providing an alternative computational paradigm to the 'program and data' von Neumann model of computation. The field of Connectionism or Neural Networks started from an interest in making models of the processes that occur in the brain. Since then the motive of modelling cognition in the brain has given way to understanding Neural Networks within the context of Statistics and Probability. Many new Neural Network architectures and training algorithms have been developed with little or no reference to biological neurons.

Neural Network research started when McCulloch and Pitts [49] in 1943 proposed a simple mathematical model of a single biological neuron. A mechanism for learning was developed later by Hebb [25] in 1949 who stated that when a neuron cell A excites another cell B then the efficiency of the excitation of cell B is increased. In the late 1950s, Rosenblatt [72, 73] developed the Perceptron in hardware. The Widrow-Hoff learning rule was also developed [92], which was based on error correction. At the end of the 1960's Minsky and Papert [53] focused on the problems of these methods, particularly their inability to solve linearly-non-separable problems. These problems were solved in the 1980's by learning algorithms based on error-backpropagation (Rumelhart, *et al.* [74], Werbos [90], Parker [60] and LeCun [39]). Another important development was the Radial Basis Function network architecture (Broomhead and Lowe [9], Moody and Darken [54]). The Multi-Layer-Perceptron (MLP) using back-propagation and the Radial Basis Function (RBF) networks are the two Neural Network architectures used for all experiments in this thesis.

Basic features of Neural Networks

Neural Networks have many features that have stimulated the rapid growth in research. They can learn from a set of patterns describing a wide range of different applications. Neural Networks are parameterised models (White [91]) that are able to approximate any function. It has been shown by Funahashi [16] that a single hidden-layer MLP can approximate any function provided the number of hidden units is large (See also Cybenko [11] and Hornik *et al.* [31]). This is because of the non-linear characteristics of the neurons. The layers of continuous activation functions give output functions that are continuous and non-linear. Neural computation offers a robust method of modelling data where faults in the individual network nodes lead to graceful degradation of performance.

Knowledge is acquired by a learning process where the knowledge is stored in the inter-neuron connection weights. The weights form the only modifiable parameters in the model. The weights and the activation functions of the network nodes form a non-linear mathematical function, which transforms a set of input variables into a set of output variables. For MLPs, learning usually takes the form of adapting the weights by an iterative process. The weights converge to values that perform the mapping described by the data set. Neural Networks are able to adapt to a non-stationary environment; this has been the main aim of techniques of incremental learning. Neural Networks are able to form generalisations on the data they are trained with. Generalisation refers to the performance of the network on patterns that were not included in the training set. Networks of interconnected neurons form a parallel structure, which can be implemented in hardware giving very fast performance (Mead and Conway [50]).

Neural Networks that approximate input-output mappings or functions are an example of supervised learning. For each input, the desired output or target is known which can form the basis of some error correcting learning process. Unsupervised learning requires a network to discover features in the data by itself. Unsupervised learning architectures include Kohonen networks [36]. Neural Networks have provided a better way of formulating mathematical models by allowing the data itself to influence the model instead of models being developed and then later validated with real data.

Neural Networks are able to perform many tasks such as regression and classification. For classification problems Neural Networks have provided a more powerful set of tools over conventional pattern classifying techniques (Huang and Lippmann [32]). There are many architectures that have been developed with these features and many innovations have been made to improve training times. The most important aim of these methods is that these networks have good generalisation ability. It will be shown that some *a priori* knowledge about the problem must form part of the model. The models must also as simple as possible to acount for the data, according to the principle of Ockham's razor.

The Efficient Reconstruction paradigm

In order for a Neural Network to learn it must be able to find a representation of the data presented. Zemel [96] describes how a learning problem can be easy to solve given a good representation system. For example a jazz soloist will be able to learn a new piece quicker if he knows about standard chord progressions and transposition rules. The soloist uses this set of features to make up the representational system. Representations can also be formed by artificial Neural Networks. The representations are formed by the strength of the weights between all the layers of nodes in the Neural Network. The hidden layers of a Neural Network are often called feature detectors or the representation layer and must form a basic representation system for the incoming data. Sometimes this layer contains fewer nodes than the number of inputs. This has the affect of forcing the network to adopt a more efficient representation in a m-dimensional space smaller than the n dimensional space of the incoming data. A Neural Network may contain many representations for individual patterns, which make up the larger representational system. The representations formed in the hidden layer of a Neural Network come about through the minimisation of some cost function, which imposes constraints on the weights and biases. The cost function can take many forms making different representations in the synaptic weights of the network. This thesis presents a new cost function with different representational properties.

The types of problems that Neural Networks are used to model fall into two categories of representation formed. A task-dependent representation relies on a representational system being known *a priori* that can be built into the model. For many problems there is little *a priori* knowledge of the underlying features of the data. In these cases the network must form the representations. A 'good representation' is one that models the structure or the true causes of the data. This has important consequences for the generalisation ability of the model to unseen data. Where the model must discover the structure for itself the representation formed is called task independent. Zemel [96] also outlines what makes a good representation. A good representation should capture the underlying structure of the data and the number of underlying features should be as small as possible. The representation must be an accurate characterisation of the observed data. The new cost function developed here is used to identify structure in data.

There are many existing methods of data compression that impose *a priori* knowledge to the encoding of data. These do not always take advantage of features perculiar to some data sets. Neural Networks allow a more data oriented approach to modelling where the representations can be learned, adapting the coding to the data. Neural Networks are a form of adaptive data compression. Methods of Vector Quantisation express this, see Gray [19]. Explanation-based learning imposes prior knowledge to a learning task. Data compression methods are useful where the data contains no noise and the data must be represented exactly. When the data is corrupted by noise the features that can be extracted from the data must not include those which will reconstruct the noise.

A test of whether the right set of features have been found in some data is to try and recreate that data using the features. This is called the efficient reconstruction paradigm where the aim is to find a concise and efficient representation of the original data. Networks that learn to reconstruct the original data are self-supervised and are called autoencoders. The new cost function will reconstruct the data expressed within a certain accuracy. Generalisation forms the basis on which Neural Network models learn to recognise patterns and refers to a network's ability to recognise patterns that are not in the training set. The network must be as simple as possible to keep the number of underlying features as small as possible. This does not explain what is meant by a simple model, however. Simple models can be described in terms of the principle of Ockham's razor. This principle expresses the idea that the system being modelled can be described with the minimum of facts. It also suggests that efficient models using as few controlling parameters as possible should be preferred over complex models. Simple models have as few free parameters as possible to account for the data. LeCun [40] confirmed this result. A well-known technique for creating simple models in Neural Networks is the process of regularisation. This is where the available parameter space that a model can have is controlled. Regularisation will be used often in this thesis to indicate structure in the data.

Minimum Description Length and simple models

Simple models can be obtained by an information-theoretic approach, which represents a model by a set of bits. Zemel [96] calls this "the number of bits in the description that some computationally effective procedure requires in order to reproduce the observations". Therefore the idea of the simplest model lends itself to being described in terms of Information Theory. Information Theory was initially developed by Shannon [78] and refers to the communication of information from a source or transmitter to a destination or receiver through a channel. The information content of a message is based on the number of choices there are in sending one message out of a set of all possible messages using a particular coding system. It is the number of bits required to express the number of choices that gives the information content of a message.

We saw earlier that a set of features can be used to recreate the data in order to test the network model. Simple models can also be defined in terms of the information required to reconstruct the outputs from the inputs. For example we may be interested in modelling the function $f(x) = x^2$. A table could be constructed which stores the output for every possible input, requiring much information and containing no indication of the underlying causes of the data. A copy of the original data set would be the most accurate representation but would not be very efficient. A more efficient model requiring less information would be to store a program implementing the function itself and use it to generate the results. A Neural Network may have a lesser information content represented in its weights, even though all the weights have to be stored. A disadvantage with efficient representations is that they may use the same representation to characterise all the data, leading to inaccuracies from the representation. This leads to a trade-off between the accuracy and efficiency of the representational system. If the data is noisy, the inaccuracy may be good in that it may not model the noise.

The Minimum Description Length (MDL) principle is a way in which Ockham's razor can be applied to models that describe a particular system. For a detailed description see Zemel [96] and Rissanen [70]. The principle is based on the following lengths.

- The length of the model.
- The length of the data when encoded using the model as a predictor.

The best model is the one that minimises the combined lengths and reconstructs the data. This principle can also be described as the information required to reconstruct the data. In this thesis a new cost function is developed that can be interpreted in terms of these principles. No attempt is made to develop a cost function that fully embodies all of this principle. The principles embodied in Information Theory and Ockham's razor are used to describe models or representations that are compact and efficient. It will also be shown that the new cost function will have the interesting property of making a Neural Network converge on a mode of a set of data during training. This will allow problems that have multi-valued mappings to be solved.

Information Theory

Information Theory was developed by Shannon [78] and refers to the communication of information from a source or transmitter to a destination or receiver through a channel. Information is described using any system of symbols or signals. A transmitter sends a message in a particular coding system. This system can be any set of symbols, for example the set of numbers 0 to 9, the alphabet or signals represented by different voltages on a wire. The information content of a message is based on the number of choices there are in sending one message out of the set of all possible messages using a particular coding system. For example the information content of the message 'a' depends on the probability of choosing any other letter from the alphabet. If each letter is equally probable in a message then the number of choices will be 26.

To compare the different information contents of messages it is useful to have a standard unit of information based on the binary system. In the example of sending a letter from the set of letters there are 26 choices, which can be represented by 5 bits. It is the number of bits required to express the number of choices that gives the information content of a message. The number of bits is easily calculated as the logarithm of the number of choices x that are equi-probable:

$$H = \log_2(x)\ . \tag{2.1}$$

Any scale of logarithm will give a measure of the information content of a message. The probability of a certain choice of message may depend on the message preceding it. A stochastic process is one in which a system produces a set of messages according to certain probabilities. For example if the English word "the" is spoken out of all the possible words then the next word is likely to be a noun.

$$\begin{aligned} H &= -[p_1 \log p_1 + p_2 \log p_2 + \ldots + p_n \log p_n] \\ &= -\sum_i p_i \log p_i, \end{aligned} \tag{2.2}$$

where $p_i, i = 1, \ldots, n$ represent the different probabilities of the individual pieces of information with arbitrary probability for the choices. These equations give a measure of the degree of randomness among the different messages and is called the Entropy of the message. If the probabilities of the different messages are the same then the Information content or Entropy is maximised. Strings of binary bits are often sent over channels representing integer values. The minimum information required to send such integers is the cost of representing the range of individual integers. The channel however may introduce noise and by sending this raw data there is no way of knowing which bits are in error. The solution is to introduce some redundancy into the coding to allow error correction, for example using parity checks. This redundancy affects the overall randomness of the message and thus affects the entropy of the message. It will lower the entropy by adding more bits. The ratio of the maximum entropy that a message could have to the entropy it does have is called the relative entropy. Speech contains much redundant information and has a relative entropy of approximately 0.5. Knowledge of the structure of speech allows us to understand noisy speech. If the form of the redundancy is known this can be used as prior knowledge that can be built into a Neural Network model.

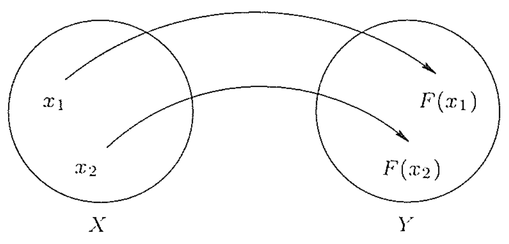

Figure 2.1: A function maps points in the range to points in the domain.

The new cost function presented in this thesis has a description length interpretation, measuring in a naive way the information required to code the data errors. The NDL cost function is naive because it does not take account of the probability distribution of the errors. Huffmann coding would find shorter codes for the most frequent errors. The NDL cost function however finds short codes for small errors.

Ill-posed problems

In supervised learning Neural Networks must be able to model the mapping from a set of inputs to a set of outputs. This is shown in Figure 2.1 where points in the domain X are mapped to the range Y with a function F. Morozov [56], Tikhonov and Arsenin [85] give three conditions for a 'well-posed' problem. The function F is 'well-posed' if the vectors in X map uniquely to distinct vectors in Y, for example two vectors $\mathbf{x}_1 = \mathbf{x}_2$ are mapped to vectors $F(\mathbf{x}_1)$ and $F(\mathbf{x}_2)$ only if $F(\mathbf{x}_1) = F(\mathbf{x}_2)$. Problems that are 'well-posed' also require that F be continuous and that for every input vector $\mathbf{x}$ in the range there exists an output $\mathbf{y} = F(\mathbf{x})$.

Most problems that are dealt with by Neural Networks are 'ill-posed'. Firstly, because the amount of data is small so that a unique mapping cannot be defined, a number of different hypersurfaces formed by the Neural Network model may fit the data. The presence of noise in the data may cause mappings to break the continuity condition for 'well-posed' problems. For a Neural Network to model such problems some prior information is needed about the mapping. This will help get over the 'ill-posed' nature of the problem (Poggio and Girosi [63]).

We have seen that a noisy signal can be corrected by the addition of redundant bits in the signal. The redundancy in the data gives vital clues to the actual underlying data. This is important in understanding speech that is subject to noise. Human speech contains a large amount of redundancy to get over this problem. Redundancy in the data can partly solve the problem of noisy data.

Prior knowledge can take many forms. It may embody the notion of 'smoothness' in an input-output interpolation problem. Many physical processes have been shown to generate data that varies smoothly, the smoothness being another example of redundancy in the data. A regularising cost function may use second derivative information of the training error surface to make sure that the network mapping is smooth (Bishop [5]). It may take the form of imposing invariances on the network, for example where Neural Networks must recognise objects at different orientations. For different applications there will be some knowledge of the environment in which it is used which can be built into the Neural Network model. Some of this prior knowledge may allow some pre-processing of the data. Regularisation is an important tool used in this Thesis to determine the presence of structure in some data.

The learning process

Neural Networks consist of a collection of synaptic weights that can be adapted to model the task. Each of the observed stimuli from the environment will affect the activity in the network of connections. The process by which these weights are adapted is determined by the learning process used and is called training. Neural Networks learn by minimising a cost function, which is a measure of how well the model fits the data. The exact meaning of a model that fits well can be built into the cost function. The new cost function presented in this thesis fits the data in a different way to the conventional sum-of-squares cost function. The new cost function interprets a good fit as one where as many points as possible are fit within a certain scale or tolerance.

Training algorithms based on 'gradient descent' of the error with respect to the weights search for a minimum of the cost function (Scales [76]). This is done by a process called back-propagation in MLP's. The networks considered in this thesis use gradient-descent methods to minimise the new cost function. Problems of gradient descent include convergence to a local minimum in network training. Stochastic algorithms have been developed to solve this problem, for example simulated annealing (Kirkpatrick *et al.* [35]).

Learning strategies can be broken down into two types, supervised and unsupervised learning. In supervised learning each stimulus from the environment has a target response, which tells the network what to do whenever it sees that input. The target responses act as an external teacher in supervised learning. Unsupervised learning does not require an external teacher and usually performs a type of feature extraction or clustering of the input data. These features may then be assigned to output nodes. This type of learning is based on assumptions of the topological and statistical properties of the data. (Becker [3], Grossberg [20], Kohonen [37], Linsker [43], Rumelhart and Zipser [75]).

Neural Networks have been applied to a variety of different problems. They can model non-linear input-output mappings of which regression problems are an example.

For example a training set may consist of input-output pairs describing a function $t = g(x)$. These tasks are supervised learning tasks where the network must adapt the weights in a network according to the desired output values in the training set. This thesis concentrates mainly on supervised learning problems.

Neural Networks are extensively used in classification tasks. The inputs in these problems represent regions, which can be labelled as belonging to a number of classes. The labelling may be known and used in training so this is again a supervised learning problem. The advantage of using Neural Networks for classification problems is that non-linear decision boundaries can be constructed between the classes. Classification problems will be described in more detail in the next section.

Another major application of Neural Networks is in time series prediction. In these problems the data contains observations of a particular system measured at fixed intervals in time, for example a set of M samples $x(n-1), x(n-2), ..., x(n-M)$. This is another example of supervised learning where the training inputs and targets are taken from the time series itself.

The learning process itself can be split into a number of different types including error correction, Hebbian and Reinforcement learning. This thesis concentrates on the use of error correction learning. In Reinforcement Learning the network is given a measure of correctness, which could be as simple as a positive or negative value indicating whether the network output was correct or incorrect (Sutton [83]). This measure is used to increase the probability of actions leading to a particular outcome if the outcome is favourable and decreasing the probability of actions if the outcome is unfavourable. Actions that do not lead to a particular outcome have their probabilities decreased.

Classification problems

Classification problems consist of data where subsets of the data that share attributes belong to a certain class. There are many ways in which the class can be labelled. The classification problems presented in this thesis use a standard MLP to solve such classification problems. A popular method of labelling class membership is the 1-of-N coding scheme where N is the number of outputs. This scheme has binary valued targets t for each class:

$$t_{kj} = \begin{cases} 1 & \text{if the input } x_j \text{ belongs to class } C_k \\ 0 & \text{if the input } x_j \text{ does not belong to class } C_k \end{cases} . \tag{2.3}$$

The outputs of the network are not guaranteed to give values that are exactly 1 for the class that the input pattern belongs to. A decision rule must be found for classifying the m outputs of the network after it has been trained. In cases where the sum-of-squares cost function is used to train networks for classification it has been shown by Richard and Lippmann [69] that the conditional expectation $E[t_k \mid \mathrm{x}]$ equals the *a posterior* class probability $P(C_k \mid \mathrm{x}), k = 1, ..., m$. The class to which the input pattern $\mathbf{x}$ belongs can be determined by finding the class with the most probability.

The statistical nature of the learning process

The learning process can be described from a statistical viewpoint; see White [91]. A set of independent variables or observations make up a vector $\mathbf{x}$. In supervised learning this maps to a dependent vector or target t. A single target is considered here; similar results apply for vectors of targets. A number of these input-output vectors make up a training set consisting of N patterns. The inputs $\mathbf{x}$ and the outputs t are related by some function $g(\mathrm{x})$ forming a regressive model. For many problems the exact functional relationship between the input vectors $\mathbf{x}$ and the target t is unknown. The dependence of the targets to the inputs is then:

$$t = g(\mathrm{x}) + \varepsilon \, . \tag{2.4}$$

The error term ε corresponds to the unknown dependence of t on x and $g(\mathrm{x})$ is a function of the vector $\mathbf{x}$. The term ε corresponds to the fact that a training set may be a subset of a larger class or rule that we know nothing about, or it may be some additive noise process. This is shown in a later section on generalisation. The function $g(\mathrm{x})$ can be defined as the conditional expectation of the value of t at a given value of the input x, $g(\mathrm{x}) = E[t \mid \mathrm{x}]$. For many problems the average value of the expected error ε is zero for all values of x,

$$E[\varepsilon \mid \mathrm{x}] = 0 \, . \tag{2.5}$$

The expected error is also uncorrelated with $g(\mathrm{x})$,

$$E[\varepsilon g(\mathrm{x})] = 0 \, . \tag{2.6}$$

The actual output of the network will be $y = F(\mathrm{x}, \mathrm{w})$ and depends on the values of the weights in the network, which will be trained by some error correction cost function. Most applications use the sum-of-squares cost function to adapt the weights, which depends on the error $e = t - y$ between the network output and the desired value t. Optimising the weights involves minimising the expected error with respect to the weights,

$$J(\mathrm{w}) = \frac{1}{2} E[(t - F(\mathrm{x}, \mathrm{w}))^2] \, . \tag{2.7}$$

Introducing $g(\mathrm{x})$ and rearranging:

$$J(\mathrm{w}) \;=\; \frac{1}{2} E\left[(t - g(\mathrm{x}) + g(\mathrm{x}) - F(\mathrm{x}, \mathrm{w}))^2\right]$$

$$= \frac{1}{2}E\left[(t-g(\mathrm{x}))^2\right] + E\left[(t-g(\mathrm{x}))(g(\mathrm{x})-F(\mathrm{x},\mathbf{w}))\right] +$$
$$\frac{1}{2}E\left[(g(\mathrm{x})-F(\mathrm{x},\mathbf{w}))^2\right] . \tag{2.8}$$

The second term is related to the expected error,

$$E[(t-g(\mathrm{x}))(g(\mathrm{x})-F(\mathrm{x},\mathbf{w}))] = E[\varepsilon g(\mathrm{x})] - E[\varepsilon F(\mathrm{x},\mathbf{w})] . \tag{2.9}$$

Both these terms in Equation 2.9 are 0 as already shown in Equations 2.5 and 2.6.

$$J(\mathbf{w}) = \frac{1}{2}E[(t-g(\mathrm{x}))^2] + \frac{1}{2}E[(g(\mathrm{x})-F(\mathrm{x},\mathbf{w}))^2] \tag{2.10}$$

The first term does not depend on the weights so a weight vector $\mathbf{w}_0$ must be found that minimises:

$$E[(g(\mathrm{x})-F(\mathrm{x},\mathbf{w}))^2] . \tag{2.11}$$

When the cost function $F(\mathrm{x},\mathbf{w})$ has been minimised to $F(\mathrm{x},\mathbf{w}_0)$ where $\mathbf{w}_0$ are the weights for the minimised cost function, the network approximates the conditional expectation $g(\mathrm{x}) = E[d \mid \mathrm{x}]$. This is the main feature of sum-of-squares cost function training.

The noise model for the sum-of-squares cost function

A Neural Network approximates the function $t = g(\mathrm{x})$. Suppose a finite data set D contains N input-output pairs $D = (\mathrm{x}_p, t_p), p = 1, ..., N$. It is assumed that additive noise v is zero mean Gaussian with variance σ for the data set. The function to be approximated becomes:

$$t_p = f(\mathrm{x}_p) + v_p . \tag{2.12}$$

It has been shown that learning is equivalent to maximising the conditional probability of the target t given the input $\mathbf{x}$. For a single input-output pattern where the sum-of-squares or Gaussian cost function is used and the output of the network is given by $f(\mathrm{x},\mathbf{w})$ the noise model is:

$$P^{(Gauss)}(t \mid \mathrm{x}) = \frac{1}{Z^{(Gauss)}} e^{-\beta E^{(Gauss)}} \tag{2.13}$$

where,

$$E^{(Gauss)} = \frac{1}{2}(f(\mathrm{x},\mathbf{w}) - t)^2 . \tag{2.14}$$

Here $Z^{(Gauss)}$ is a normalising factor and β expresses the noise on the targets.

The conditional probability of the target given the network input is also called the likelihood for the target t. The data points are independently drawn from the probability distribution describing the inputs, $p(\mathrm{x})$. The likelihood of the complete

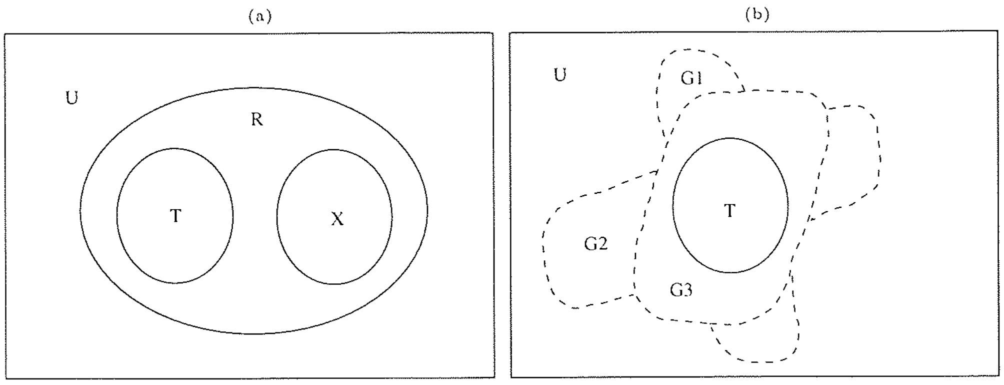

Figure 2.2: Neural Networks can form different generalisations on a test set. (a) The training and test set are part of the set R. (b) The different valid generalisations that the network can make.

data set of targets given the set of inputs is the product of the individual likelihoods. The product of exponentials of terms is equivalent to the exponential of the sum of the terms. This means that the likelihood of all the targets given all the inputs or the probability of the data set is:

$$P^{(Gauss)}(T \mid X) = \frac{1}{Z^{(Gauss)}} \prod_p e^{-\beta E^{(Gauss)}} = \frac{1}{Z^{(Gauss)}} e^{-2\beta E_D} \tag{2.15}$$

where E_D is,

$$E_D = \frac{1}{2} \sum_{p=1}^{N} [f(\mathbf{x}_p, w) - t_p]^2 , \tag{2.16}$$

$$Z^{(Gauss)} = \left(\frac{2\pi}{\beta}\right)^{N/2} . \tag{2.17}$$

Here $\beta = 1/\sigma_\mu^2$ is the noise on the targets.

Generalisation

This thesis will present networks that will generalise in a different way to networks trained with the conventional sum-of-squares cost function. These networks will form solutions that give very different generalisations to unseen data, each of which may be equally valid. Denker *et al.* [12] have shown how networks can give different generalisations after being trained with a small subset T of all the possible input-output pairs U; see Figure 2.2a (from Hertz [27] and Denker [12]). The set T is a sub-set of the set R that contains the sets T and X. The network must generalise to give results consistent with membership of the set R, when tested with the set X. Figure 2.2b shows a collection of possible generalisations that are consistent with the set T. Many such sets could be given for networks trained with different initial weights, some will be consistent with the set R and some not.

The number of different generalisations can be shown for a simple example. Suppose a particular input vector contains N binary digits, which can give 2^N possible input patterns. For each of these the output can have the value of 0 or 1, thus making the total number of patterns 2^{2^N}. In practical applications the number of generalisations is reduced because a network may not be flexible enough to capture all generalisations. Networks with large numbers of nodes would give a greater number of possible generalisations.

If each of the possible input patterns is written in a truth table then if p of these patterns make up a training set then p of the 2^N combinations will be set. This leaves 2^{2^N-p} combinations that a network could validly generalise to. This shows that generalisation can be quantified by the number of different generalisations it can make from the training set (More details can be found in Hertz [27]).

The idea of a number of different generalisations can be expressed by the average generalisation ability of a Neural Network model. The following discussion can be applied to data that does not contain noise. The test set X was chosen from a different part of the input space than the training set T. For most problems the test set is drawn from the same probability distribution $p(\mathbf{x})$ as the training set. Schwartz *et al.* [77] discussed a theoretical framework for calculating the average probability of correct generalisation for a training set of size p.

Suppose we have a network with a fixed architecture. It will have a set of weights $\mathbf{w}$ out of the set of all possible vectors called a weight space. This weight space may be restricted by some *a priori* probability density on the weights. For example the weight values may be drawn from a uniform probability distribution between 0 and 1. The total volume V_0 of the weight space is then:

$$V_0 = \int p(\mathbf{w})\, d\mathbf{w}\ , \tag{2.18}$$

where $p(\mathbf{w})$ is the a *priori* probability density of the weights. Each weight vector in weight space will implement a certain function $f_{\mathbf{w}}$ mapping the inputs $\mathbf{x}$ to the network outputs y. The function is assumed to have the form $F : R^n \to \{0, 1\}$. The same principles can be applied to continuous-valued outputs. Some of the weight vectors will implement the same function. Therefore a particular function f in Figure 2.3a represented by the weights will occupy a region of the weight space:

$$V_0(f) = \int p(\mathbf{w})\Theta_f(\mathbf{w})\, d\mathbf{w} \tag{2.19}$$

where

$$\Theta_f(\mathbf{w}) = \begin{cases} 1 & \text{if } f_{\mathbf{w}}(\mathbf{x}) = f(\mathbf{x}) \quad \text{for all } \mathbf{x} \\ 0 & \text{otherwise} \end{cases} . \tag{2.20}$$

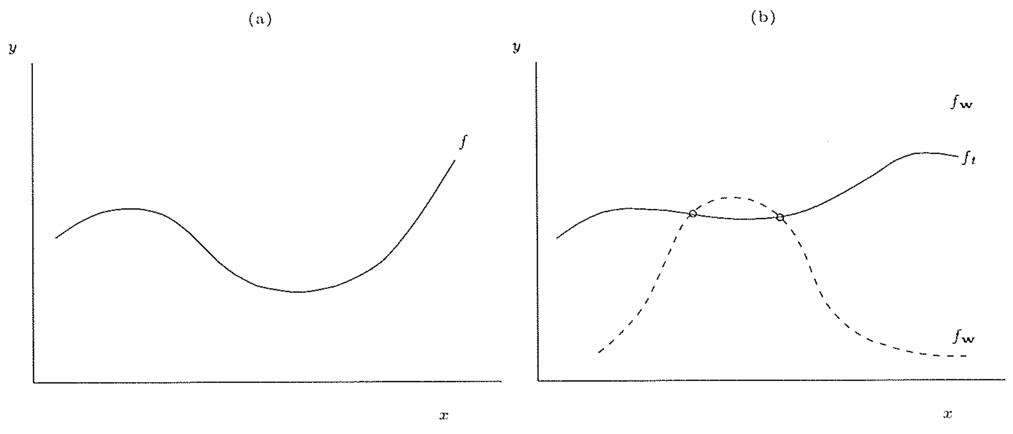

Figure 2.3: Two networks different weight regions. (a) A function f will have a set of weights that implement that function. Other weight vectors may implement the same function. (b). The desired function f_t and a couple of data points. Many different functions will be consistent with the data set.

Normally a number of training patterns (X, T) will be presented to the network expressing a particular desired mapping f_t where $X = \{\mathbf{x}_1, ..., \mathbf{x}_p\}$ and $T = \{t_1, ..., t_p\}$. If p patterns are learned successfully then the weight vector $\mathbf{w}$ will lie in a region of weight space consistent with these patterns. Re-expressing Equation 2.18 for the number of patterns p gives:

$$V_p = \int p(\mathbf{w}) \prod_{k=1}^{p} I(f_\mathbf{w}, \mathbf{x}^k)\, d\mathbf{w} \,, \tag{2.21}$$

$$I(f_\mathbf{w}, \mathbf{x}^k) = \begin{cases} 1 & \text{if } f_\mathbf{w}(\mathbf{x}^k) = f_t(\mathbf{x}^k) \\ 0 & \text{otherwise} \end{cases} \,. \tag{2.22}$$

The region V_p includes the region of the weight space that gives the target function f_t as well as other regions giving functions that agree with f_t on the training set, see Figure 2.3.b As the number of training patterns p increases the value V_p is reduced as less of the weight space is used to model the greater number of patterns. The likelihood of other functions matching all these patterns decreases. V_p will exclude more alternative functions f.

The volume of weight space that is consistent with both the function f and the training patterns is given by:

$$\begin{aligned} V_p(f) &= \int p(\mathbf{w}) \Theta_f(\mathbf{w}) \prod_{k=1}^{p} I(f_\mathbf{w}, \mathbf{x}^k)\, d\mathbf{w} \\ &= V_0(f) \prod_{k=1}^{p} I(f, \mathbf{x}^k) \,. \end{aligned} \tag{2.23}$$

The $I(f, \mathbf{x}^k)$ can be taken out of the integral because $\Theta_f(\mathbf{w})$ means $f_{\mathbf{w}}$ is equal to f in all non-vanishing cases. If f agrees with the training patterns then $V_p(f)$ is equal to $V_0(f)$. The training patterns themselves can be assumed to have been drawn from a distribution $p(\mathbf{x})$ of all possible input patterns. Averaging over all possible training sequences gives a measure of the generalisation ability:

$$g(f) = \langle I(f, \mathbf{x}) \rangle = \text{Prob}(f(\mathbf{x}) = f_t(\mathbf{x})) \,. \tag{2.24}$$

The term $g(f)$ is called the generalisation ability of f and is the probability of a function f in weight space of all functions that agree with the function f_t described from the limited number of patterns.

Regularisation

Prior information such as the requirement that the functions being modelled show a degree of smoothness can be introduced into the model of the data generator. This can be done by adding a cost function that contains the prior information to the learning process. This function can be added to the standard error function. This forms the basis of regularisation. Regularisation Theory was proposed by Tikhonov [85] and involves the calculation of these two terms. The first term measures the error between the actual response y_i of a network and the target or desired response t_i for a number of training patterns N. This error term usually takes the form of the sum-of-squares error between the output y_i and the target response t_i:

$$W_e(F) = \frac{1}{2}\sum_{i=1}^{N}(t_i - F(x_i))^2 \,. \tag{2.25}$$

The second term is the regularising term,

$$W_r(F) = \frac{1}{2}||\mathbf{P}F||^2 \,. \tag{2.26}$$

where $\mathbf{P}$ is a differential operator that contains the prior knowledge that we wish to build into the model. The sum of these two terms gives the cost functional that must be minimised,

$$W(F) = W_e(F) + \alpha W_r(F) \,. \tag{2.27}$$

The α parameter is called the regularisation parameter and controls the relative importance of the two terms. When the value of α approaches zero this means that the network model is unconstrained and must be formed from the data alone. As the value of α approaches infinity, then the prior knowledge is the important term and can be regarded as characterising the data alone. Networks must be found that will incorporate both these terms to some greater or lesser degree.

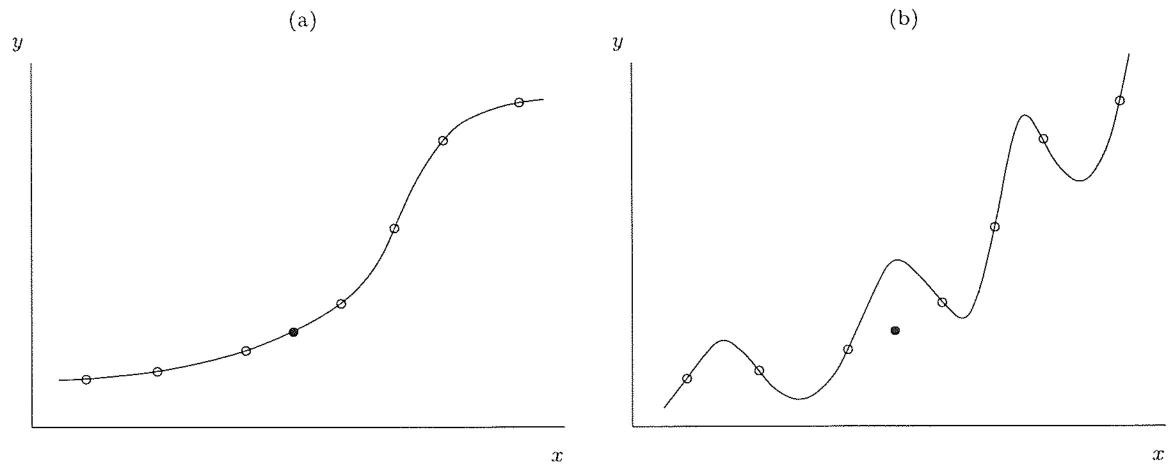

Figure 2.4: A set of points can be modelled by different polynomials. (a) A polynomial with a few coefficients. (b) A polynomial with many coefficients. A test point shown as a black circle lies along the smoother curve with small error.

Generalisation and curve fitting

The relation between a model selection criterion such as Ockham's razor (Blumer *et al.* [8]) and the smoothness of the input-output mapping was pointed out by Poggio and Girosi [63]. When prior knowledge is not available the simplest function is selected, which is important for good generalisation.

Generalisation can be viewed as a curve fitting problem. The conventional method of performing interpolation based on a set of points is to fit a polynomial to the points. Interpolation refers to a model's ability to model the function described by a set of input-output vectors and estimating the output for inputs not used in training. Figure 2.4 shows how two polynomials with different numbers of coefficients can fit the data. Extrapolation refers to estimating the function outside the range of values used in training. The coefficients in the polynomial are analogous to the weights in a Neural Network. The more coefficients the polynomial has the more 'complex' it is. Generalisation can be viewed as finding a good non-linear interpolation of the input data (Wieland and Leighton [93]). Figure 2.4 can also show the different generalisations made by a Neural Network. The training patterns are used to approximate a function given by the solid curves in the graphs. Prior knowledge of smoothness would favour Figure 2.4a over 2.4b as a better model for the data.

The generalisation ability of a Neural Network is determined by the number of patterns used in training. The complexity of the problem also influences the generalisation capabilities of a network model. Hush and Horne [33] consider the issue of determining the size of the training set needed to achieve good generalisation if the network archi-

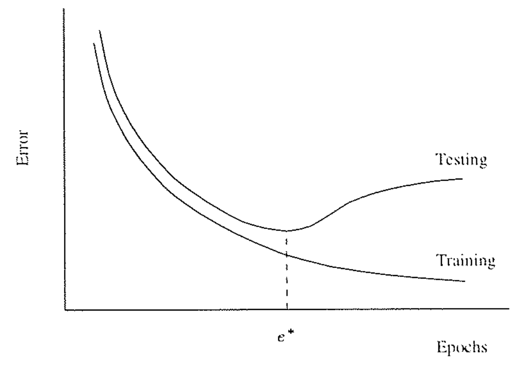

Figure 2.5: The training and test error against the number of iterations or training epochs. The generalisation or test error reaches a minimum at e^*.

tecture is fixed. They also consider a fixed training set for which an optimum network architecture must be found to achieve good generalisation.

Some theory on the size of the training set required to give good generalisation was done by Baum and Haussler [2]. They considered a data set with no noise consisting of N training patterns with W weights and M hidden nodes. Good generalisation can be achieved provided the number of training set patterns exceeds a certain limit where ε is the fraction of errors allowed on a test:

$$N \geq \frac{32W}{\varepsilon} \ln\left(\frac{32M}{\varepsilon}\right) . \tag{2.28}$$

The fraction of training set errors must also be less than $\varepsilon/2$. The practical considerations of this method are discussed by Haykin [24].

Cross-validation

One popular way of achieving a network with good generalisation capability is by validation of the network during training (Morgan and Bourlard [55] and Hergert *el al.* [26]). The validation or generalisation error decreases monotonically as training continues. After a set number of training epochs the network performance is tested. For most problems the generalisation error for a test set will reach a minimum and then increase; this is shown in Figure 2.5. The interpolation and extrapolation capabilities of the network may reach a maximum when the network has not fully fit all the data. An explanation of this behaviour is given by Weigend *et al.* [89]. They explain that at the start of training the effective number of free parameters or weights are small even though a large network is used. This is reflected in the initial values of the weights. The effective number of free parameters increases until there is just the right number to correctly model the data. Wang *et al.* [86] also argue that a critical region exists in training where generalisation is at its best. Stopping training just before the global minimum has the effect of selecting the network complexity that will model the data.

The size of the original data set is important to finding a network with good generalisation capability. For many problems a training and test data set can be created with the number of patterns much greater than the number of free parameters or weights in the network. Where data sets are small the available data must be partitioned sensibly into a training and testing set. A method has been developed to deal with this particular problem called cross-validation, see [81, 82] and Janssen [34]. The test set itself is used to optimise the performance of a network. An independent validation set is needed to check the final performance on the trained network.

Cross-validation splits up a data set randomly into N equal sized blocks. The network is trained on $N - 1$ of the blocks and its performance is measured on the remaining block. The networks is trained again on the other $N - 1$ blocks giving N possible results. The test set results are then averaged. This method is a good one if the size of the data set is small. The average error can then be compared with the average error from other networks. Cross-validation is a method of choosing between networks that may have completely different architectures and trained with different hyper-parameters like α. The whole training set is then used when the best architecture or parameter is selected.

Pre-processing

When networks are used in real-life applications the performance can be poor if the raw data is used. Pre-processing of the inputs and outputs and the use of the correct representation can greatly improve generalisation. Certain representations may not capture the essential features of the data. There is a class of problems for which Neural Networks cannot form the correct representations, these include incomplete training sets for binary problems (Stone [80]). It has already been shown that Neural Networks can model any function; however some problems can benefit from pre-processing and post-processing to provide some actual physical quantity as outputs of an application. Pre-processing can do some basic transformations that reduce the dimensionality of the incoming data. For some of the experiments presented in this thesis, the dimensionality of the problem was deliberately increased by using coarse-coding. This was done to make sure that overfitting of the data occurred for very small amounts of regularisation. In this way the different levels of structure could be seen by examining the dependence of the error on the value of the regularisation parameter α.

Pruning and Constructive algorithms

Another method of achieving good generalisation for an MLP network is to control the network complexity by controlling its architecture. The minimal size parameter set must be found that will model the data (Hanson and Solamon [22] and Smith [79]). A network that is small with a few nodes will not be capable of overfitting the noise in a particular problem. Too many nodes will lead to overfitting, which may not be optimal because of the 'ill-posed' nature of the finite data set. This is true for clean as well as noisy data. There are two methods of achieving the optimum network architecture. Some algorithms, called constructive algorithms, start with a network with a few nodes and attempt to grow the network. Nodes are added to the network until the network performance reaches an acceptable value. A good example of this technique is cascade-correlation (Falhman and Lebiere [14]). Another algorithm by Lee [42] adds neurons when the error exceeds a certain value. It also places nodes where they are most needed by examining the variation in value of neighbouring weights. A mapping may be better represented by the addition of a node to the network. The variation in value or the sensitivity of the outputs with respect to the weights is used in the approach by Wynne-Jones [95] where nodes are split to add more representational capability to a network. Other constructive algorithms include Upstart by Frean [15], the Tiling Algorithm (Mezard [51]) for linearly-separable problems involving binary input patterns and the Pocket algorithm of Gallant [17]. Other algorithms are given by Ash [1] and Refenes [68]. There is also the Forward Selection method used by statisticians.

Once a network has been trained to model a particular function, generalisation can be improved by optimising the number of free parameters or weights in the network. Pruning methods include those which identify nodes or weights that have no effect on the mapping being modelled. Other methods attempt to drive weights in a network to zero. This occurs in weight elimination due to Weigend *et al.* [89]. For a description of different pruning algorithms see Reed [67]. Weight decay (Plaut *et al.* [62] and Hinton [28]) was one of the first techniques used to reduce the effective number of free parameters in a network. Weight decay penalises weights in proportion to their magnitude and the improved generalisation ability of feedforward networks was confirmed by empirical studies by Hinton [29]. Krogh and Hertz [38] give a theoretical justification of better generalisation using weight decay. Other methods involving weight-sharing have been developed by Nowlan and Hinton [57, 58]. More methods involving weight elimination are found in Hanson and Pratt [23], Chauvin [10], Thodberg [84]. There is also Backward Elimination used by statisticians (Rawlings [66]).

Information-theoretic approaches to controlling network size and complexity include the method known as Optimal Brain Damage (LeCun *et al.* [41]). With this method second-derivative information of the error surface is preferred to measure a weight's 'saliency' over other methods where saliency is measured from the magnitude of the weights. The second derivative information is used to derive a measure of the network's information content to trade-off network complexity with training set error. It reduces the size of the network by selectively deleting weights.

Hanson [23] has observed that standard back-propagation makes use of all the hidden units in a network, the representation being distributed over all the weights, even for oversized networks. The weight decay penalty term does not improve matters where the representation can be made by two equal weights instead of by a single weight with the other eliminated. The objective, as with weight elimination, is to drive small weights to zero without penalising large weights that can model the data. Experiments have been done that show that constraining subsets of weights to share similar values can lead to improved generalisation. Weight-sharing uses mixtures of Gaussians to model the weights. The means and variances of these Gaussians are adapted as the network is trained. A similar approach is made by Hochreiter [30] where the weight space is partitioned into 'boxes' that give regions of weight space where the error remains approximately constant. These regions are called 'flat minima'.

Dimensionality reduction

Another approach to achieving minimal networks is by doing some analysis of the data itself to see if the amount of data used in training can be reduced by removing redundancy in the data. It may be possible to make abstractions that will greatly simplify the description of the data. Some abstractions are very necessary to build prior knowledge into the description of the data. This is so that a Neural Network can generalise from incomplete training sets for certain problems (Stone [80]). Any method that removes redundancy and captures the natural degrees of freedom in the data set will give a more succinct description of the data. However, one must be careful with the removal of redundant data when it takes part in error correction as the example of speech has shown.

The data set may have a number of input vectors with n inputs or features. The vectors will lie in a n-dimensional space. These vectors may lie on a m-dimensional surface embedded in the n-dimensional feature space. The amount of information needed to code such a set of features will be reduced if each data point is specified by a m-dimensional location on the embedded surface. The procedure for finding such embedded surfaces is called dimensionality reduction.

Many techniques exist for data compression, of which Principal Component Analysis (PCA) is one. Principal Component Analysis is a linear method of data compression.

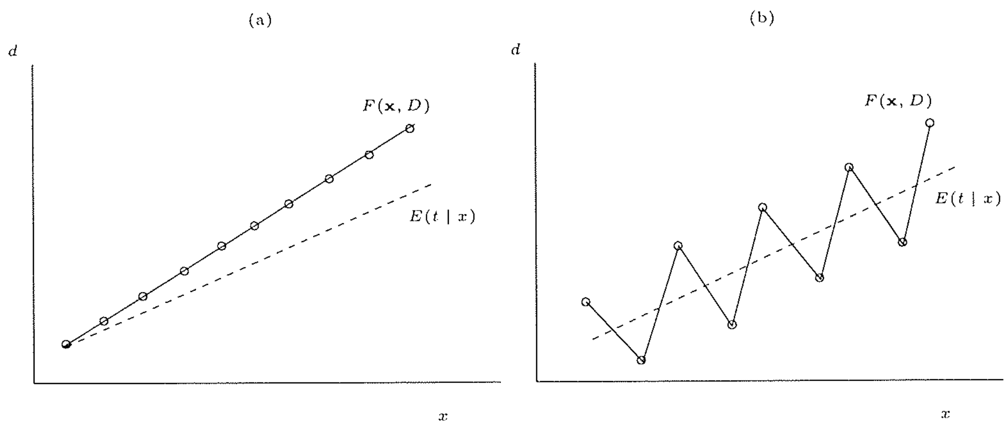

Figure 2.6: Two networks showing (a) bias and (b) variance.

PCA can find a set of features in terms of variances of different linear combinations in the data. PCA performs a form of feature extraction where the input data is transformed into a feature space with the same dimensionality as the original data. Another transformation takes account of the most significant features, discarding other features that do not result in a significant loss of information in the data. Principal Components involves the calculation of the eigenvalues and eigenvectors of the correlation matrix of the input data. The data is projected orthogonally onto a subspace according to the first m principal components where $m < n$, where n is the total number of components or the dimensionality of the input data. In this way the number of features needed to accurately represent the data is reduced by discarding linear combinations with low variances (Oja [59]). In many cases the variances approach zero so the data transformation results in very slight error. This technique is called subspace decomposition. Neural Networks may be forced to make succinct representations through dimensionality reduction when the hidden layer has fewer units than the input, causing a 'bottleneck'.

The Bias/Variance dilemma

Neural Networks such as the feed-forward type used in this thesis can be described as a form of non-parametric statistical inference. They must estimate the decision boundaries in classification problems. They must also form regressive models from data sets describing mappings from a set of inputs to a set of outputs. It has been shown that the behaviour of Neural Networks can be described from a statistical framework. This framework also allows us to examine the estimation error from a statistical viewpoint. The estimation error or the mis-match between the outputs of the Neural Network and the actual generator of the data is made up of two components, the bias and the variance. Model-free inference allows a network to describe the data from the data alone with little of no prior knowledge. This tends to give networks that suffer from high variance. Large training sets are needed to keep the variance low to make up for the absence of prior knowledge. The variance can also be reduced by introducing prior knowledge or by using a model-based approach. However these models are unlikely to model the data exactly and so can lead to high bias. For some complex tasks the data sets may be small, leading to high variance.

The estimation error between the data generator and the output of a single output Neural Network can be expressed as:

$$(g(\mathbf{x}) - F(\mathbf{x}, \mathbf{w}))^2 = (E[t \mid \mathbf{x}] - F(\mathbf{x}, \mathbf{w}))^2 . \tag{2.29}$$

A network is trained with a finite data set $D = (\mathbf{x}^k, t^k)$, $k = 1$ to N. This data set must be represented in the weights of the network so $F(\mathbf{x}, \mathbf{w})$ is rewritten as $F(\mathbf{x}, D)$. The expectation E is rewritten as E_D as it represents the average over all training sets D. By introducing the term $E_D[F(\mathbf{x}, D)]$ and rearranging, Equation 2.29 becomes:

$$\begin{aligned} E_D[(E[t \mid \mathbf{x}] - F(\mathbf{x}, D))^2] &= (E_D[F(\mathbf{x}, D)] - E[t \mid \mathbf{x}])^2 + \\ &\quad E_D[(F(\mathbf{x}, D) - E_D[F(\mathbf{x}, D)])^2] . \end{aligned} \tag{2.30}$$

The two terms of Equation 2.30 are called the bias and variance by Gemen *et al.* [18]. If, on average $F(\mathbf{x}, D)$ is different from $E[t \mid \mathbf{x}]$ in $(E_D[F(\mathbf{x}, D)] - E[t \mid \mathbf{x}])^2$ then $F(\mathbf{x}, D)$ is a biased estimator. An unbiased estimator may still have a large sum-of-squares error if the variance is high. This can occur when $F(\mathbf{x}, D)$ is different to $E[F(\mathbf{x}, D)]$ for $E[F(\mathbf{x}, D)] = g(x)$. Incorrect model-based networks have a high bias. Model free networks have high variance. Bias and variance can only be eliminated when the size of the training set becomes very large. Bias can be built into the network as prior knowledge by constraining the network architecture, satisfying one of the conditions for turning an 'ill-posed' problem into a 'well-posed' one. Figure 2.6 shows examples of networks with bias and variance.

Extrapolation and novel data

It has been shown that Neural Networks can be used for interpolation problems. Obtaining good generalisation for input data that is very different from the input patterns used in the training set is more difficult. Neural Networks must continue to generate reliable outputs once it is operating within a particular environment. However in some industrial processes the network must be able to cope with random and possibly catastrophic events. This problem has been a major issue in the acceptance of Neural Networks by the commercial world. If an input exists outside a particular domain some

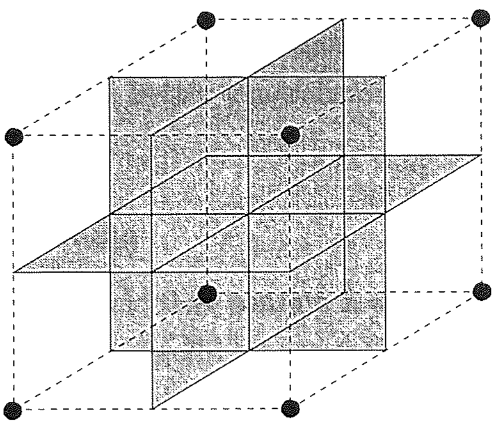

Figure 2.7: A 3-dimensional hypercube showing how a 3-input feature space can be segmented into 8 smaller hypercubes.

measure of confidence must be placed in the network outputs. When new input data is novel the confidence placed in the network output must be low because of the inverse relationship between novelty of the input data and the validity of network outputs.

Bishop [7] shows that the approximation of the data generator will be more accurate in regions of the input space where the density $p(\mathbf{x})$ is high. In regions where the density of $p(\mathbf{x})$ is small a new point will most likely be a novel point. It has been shown by Qazaz [64] that error-bars can be placed on the network outputs which depend on the density of the inputs.

Curse of dimensionality

Problems of generalisation may occur because the number of attributes or dimensionality of the inputs in the problem is large. A well-known problem in training Neural Networks is the curse of dimensionality (Bellman [4]). Figure 2.7 shows a 3-dimensional unit hypercube. Each dimension can be split into N segments, in this case two, to differentiate between the possible values in that dimension. As the number of segments increases each value can be specified more precisely. For a 3-dimensional input this gives 8 (N^d where d is the dimension of the input) small hypercubes describing all the possible values in the inputs. Therefore, N^d patterns have to be provided to specify a mapping from a set of inputs to a single output. As the number of dimensions increases the amount of data needed grows exponentially; this is the curse of dimensionality.

In most practical applications this is not such a problem because of correlations in the inputs that reduce the dimensionality. It has been found paradoxically that generalisation ability can improve when data is taken out of a training set. Dimensionality reduction is a technique for reducing the amount of data presented to a network. This reduction can take place as a part of network training or by careful pre-processing if *a priori* knowledge of the problem allows the amount of training data to be reduced. Reducing a training set may give better generalisation even though some information may be lost.

The Bayesian view of learning

Ockham's razor posits that models that contain few controlling parameters are to be preferred over ones with many controlling parameters. A complex model with many parameters, for example a network with many hidden nodes, can represent a large range of functions. They can describe a large range of data sets. A simple model with few parameters can represent a small range of functions. They can describe only a small range of data sets. This principle is embodied within the Bayesian perspective of data modelling. Bayesian methods of inference have increasingly been used for problems where the amount of data available is small (for a detailed comparison of Bayesian methods with conventional statistical techniques see Loredo [44]). Bayesian probability theory is a simple mathematical language for quantifying the plausibility of a model.

For training Neural Networks there is an uncertainty associated with how well a network models the true generator of the data. Take a 1-dimensional problem, for example a number of points along the x axis. Such a sample of points can be described by a probability distribution. In this case we can imagine a generator of random samples G. This random number generator produces random numbers according to a particular probability distribution $P(\mathrm{x} \mid G)$. Suppose that we want to find out the particular probability distribution of G but we do not know what it is. The aim is to hypothesise a model that is a good estimate of G. This hypothesis will be called $\mathcal{H}$. The precise definition of a model is not specified. It can mean an MLP with a particular set of weights and a couple of adjustable parameters. In this case the architecture is fixed and networks with different weights represent different models,

$$P(\mathrm{x} \mid \mathcal{H}) \approx P(\mathrm{x} \mid G)\,. \tag{2.31}$$

But all we really have is some data produced by G,

$$D = \mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3, \mathrm{x}_4, \ldots \tag{2.32}$$

Bayesian probability theory allows inferences to be made from finite samples, where the plausibility of a particular model is adapted when the knowledge of the model changes due to the arrival of data. Bayes' Theorem gives a probabilistic interpretation to learning. Bayes' Theorem is defined as:

$$P(\mathcal{H} \mid D) = \frac{P(D \mid \mathcal{H})P(\mathcal{H})}{P(D)} \tag{2.33}$$

or,

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}} \,. \tag{2.34}$$

The probability of the model is the joint probability of the weights and all its variables. This represents the prior probability of the model. Bayes' Theorem can be used to find the most probable weights to approximate a particular function. Bayesian methods have led to three levels of inference in the task of data modelling (Mackay [47, 46]). Bayes' Theorem is used in different ways to achieve modelling of the data given a fixed architecture, selecting the hyper-parameters of the model with the fixed architecture and model selection of different architectures.

Fitting the model to the data

A model, not necessarily a Neural Network, containing a set of free parameters is fit to the data. This forms the first level of inference and finds the most probable parameters of a model $\mathcal{H}$ given the data. Bayes' Theorem gives the posterior distribution of the weights given the data,

$$P(\mathbf{w} \mid D, \alpha, \beta, \mathcal{H}) = \frac{P(D \mid \mathbf{w}, \beta, \mathcal{H}) P(\mathbf{w} \mid \alpha, \mathcal{H})}{P(D \mid \beta, \alpha, \mathcal{H})} \,. \tag{2.35}$$

For a Neural Network the hypothesis or model takes the form of a set of weights. These weights have a prior probability distribution before the data arrives. It is usually assumed to be Gaussian. Where E_W is weight-decay regularisation and α is the regularising parameter the prior becomes:

$$P(\mathbf{w} \mid \alpha, \mathcal{H}) = \frac{e^{-\alpha E_W}}{Z_W} \,. \tag{2.36}$$

The probability of the data given the hypothesis of the model is assumed to be related to the error by a zero mean Gaussian,

$$P(D \mid \mathbf{w}, \beta, \mathcal{H}) = \frac{1}{Z_D} e^{-\beta E_D} \tag{2.37}$$

where E_D is the sum-of-squares cost function for all patterns used for training. The term Z_D is a normalising constant and β is a term that represents the noise on the targets. The cost function is also called the log likelihood.

The aim is to find the maximum of the posterior, giving the most probable weights given the data. The evidence forms a normalising constant, which is ignored in this level of inference.

$$P(\mathbf{w} \mid D, \beta, \alpha, \mathcal{H}) = \frac{e^{-M(\mathbf{w})}}{Z_M(\alpha, \beta)} \tag{2.38}$$

where,

$$M = \beta E_D + \alpha E_W \tag{2.39}$$

$$Z_M(\alpha, \beta) = \int e^{-M} d\mathbf{w} \,. \tag{2.40}$$

Finding the maximum of the posterior is equivalent to finding the minimum of M. Equation 2.39 shows how the cost function and regularising term can be understood from a Bayesian viewpoint. The cost function can be minimised using standard gradient descent techniques.

Finding the values of α and β

In the same way that Bayes' Theorem can be used to determine the most probable weights, it can also be used to find the most probable values of the hyper-parameters α and β,

$$P(\alpha, \beta \mid D, \mathcal{H}) = \frac{P(D \mid \alpha, \beta, \mathcal{H}) P(\alpha, \beta \mid \mathcal{H})}{P(D \mid \mathcal{H})} \,. \tag{2.41}$$

The quantity $P(D \mid \alpha, \beta, \mathcal{H})$ is called the evidence for α and β and gives the probability of the data given particular values of α and β. The hyper-parameter α is the regularisation parameter and must be chosen to prevent fitting the noise and allow the network to model the real function. The prior $P(\alpha, \beta \mid \mathcal{H})$ is assumed to be flat and the normalising factor $P(D \mid \mathcal{H})$ is ignored so the posterior becomes proportional to the evidence. The evidence can also be calculated as:

$$P(D \mid \alpha, \beta, \mathcal{H}) = \int P(D \mid \mathbf{w}, \beta, \mathcal{H}) P(\mathbf{w} \mid \alpha, \mathcal{H}) d\mathbf{w} \,. \tag{2.42}$$

The posterior can be defined with the normalising constants already defined,

$$P(D \mid \alpha, \beta, \mathcal{H}) = \frac{Z_M}{Z_W Z_D} \,. \tag{2.43}$$

The posterior must be maximised. Maximising the posterior is equivalent to minimising the logarithm of $P(D \mid \alpha, \beta, \mathcal{H})$. Differentiating with respect to α and β and finding the minimum gives (Mackay [46]):

$$\alpha = \frac{k - \gamma}{2E_W} \,, \tag{2.44}$$

$$\beta = \frac{N - \gamma}{2E_D} \,. \tag{2.45}$$

The term γ measures the number of effective weights used for a particular set of weights and is always smaller than the number of weights in the network k. When α is large the number of effective weights tends to zero. Training a network involves gradient descent of the error surface with an arbitrary value of α. The value of α is then selected using the above technique and training continues. This cycle is repeated until the network converges to a minimum error. This procedure is however controversial, see Wolpert [94] and Mackay [48].

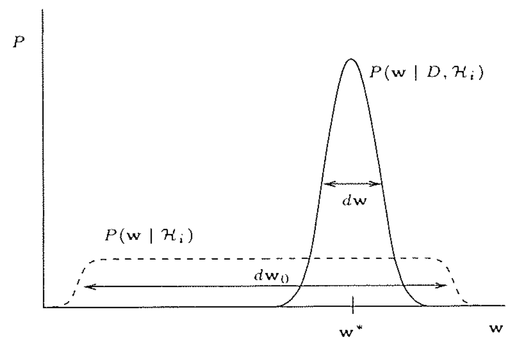

Figure 2.8: When the data arrives the distribution of the weights collapses. The ratio of the prior weights distribution width $d\mathbf{w}_0$ and the posterior weights distribution width $d\mathbf{w}$ gives the Ockham factor.

Ockham Factor and Bayes

Different models can be ranked by comparing the different evidences if each model is equally likely. The evidence $P(D \mid \mathcal{H}_i)$ can indicate the extent to which the posterior collapses when the data arrives,

$$P(\mathcal{H}_i \mid D) = \frac{P(D \mid \mathcal{H}_i)P(\mathcal{H}_i)}{P(D)} \,. \tag{2.46}$$

If a network model is chosen after training with weights $\mathbf{w}^*$ corresponding to the most likely weights of the best fit likelihood then

$$P(\mathbf{w} \mid D, \mathcal{H}_i) \propto P(D \mid \mathbf{w}^*, \mathcal{H}_i)P(\mathbf{w}^* \mid \mathcal{H}_i)d\mathbf{w} \tag{2.47}$$

where $d\mathbf{w}$ is the width of the distribution $P(\mathbf{w}^* \mid D, \mathcal{H}_i)$ centred at the most probable weights $\mathbf{w}^*$. Before the data arrives the prior $P(\mathbf{w} \mid \mathcal{H}_i)$ will be broader with a width $d\mathbf{w}_0$. The Ockham factor is the ratio of these two widths $d\mathbf{w}/d\mathbf{w}_0$.

The Ockham factor is the ratio of the posterior uncertainty in the weights to the prior uncertainty. Before any data arrives the prior will be flat as shown in Figure 2.8. A complex model with many weights and parameters can fit many data sets. The arrival of data will cause the weight distribution to collapse. This is because only a few parameter values can model the data accurately. The ratio $d\mathbf{w}/d\mathbf{w}_0$ will be very small. For simple models with a few weights and parameters, the arrival of data will cause a smaller collapse in the weight distribution. The simpler model will have a larger set of weight vectors that can model the data. This again shows the trade-off between simple models that underfit the data and complex models which overfit the data.

The L-curve

Another method of selecting model parameters like the regularising parameter α is by using the L-curve of Hansen and O'Leary [21]. The L-curve is a plot of the size of the regularised solution $\eta(\alpha)$ against the network error $\rho(\alpha)$. This is calculated for a range of values of the regularising parameter α. The curve has a distinctive 'L' shape. The corner of the 'L' indicates where the network solution changes from being dominated by the regularisation errors leading to underfitting to a network solution dominated by the errors leading to overfitting. The size of the network solution is defined in Hansen and O'Leary [21].

Chapter 3

Regularisation, Generalisation and NDL

3.1 Multi-valued mappings

Neural Networks have been used to learn single-valued mappings, for example learning to recognise the inputs to an XOR function and giving the appropriate output values. This thesis discusses a cost function that can deal with the problem of multi-valued mappings where for each individual input there may be a number of possible outputs. It is also an aim to show that this new cost function can still deal with the type of problems conventional Neural Networks are used for where the data is not multi-valued.

It may be that the incoming data has a clearly distinguishable noise-data structure as illustrated in Figure 3.1. The data that we are interested in lies along line 1 but due to noise a small number of the data points lie along line 2. In this case a Multi Layer Perceptron (MLP) [75], may converge on 3, which is the mean fit of the data and the noise. The preferred fit is along line 1, which does not take account of the noise along line 2. The cost function must recognise where the majority of the data points lie and ignore other points outside this region. Other input patterns may exhibit this kind of structure. Alternatively, the data in Figure 3.1 may be multi-modal where a network fitting the points along line 1 or 2 would be equally valid.

An example of multi-modal data is in speech where for each phoneme there are a number of different ways of expressing that phoneme acoustically. This acoustic variation follows a probability distribution and is unlike conventional training data, which is concentrated on single-valued mappings. Here it may be preferred that a network converges on the mode of the probability distribution where the data is the most probable rather than the mean. For many problems the mode may equal the mean. Conventionally each phoneme has been described by a Gaussian probability distribution and

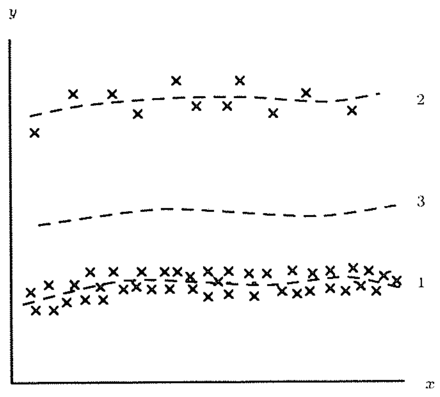

Figure 3.1: Data and noise: 1 represents the required interpolant fitting the most probable data. 2 represents an interpolant fitting the noise or may be a branch of a multi-valued mapping. 3 is the mean obtained by using conventional MLP's.

the probability of the acoustic properties (shown by speech spectrograms) describing a particular phoneme depends on this probability distribution. The communication of speech also contains much noise so it would be desirable to try and filter out this noise. The data that we are interested in will generally have a pattern and be deterministic. The noise will be random or non-deterministic. Though the noise is random it will have some structure if it can be described by a probability distribution. The largest mode of this distribution could be viewed as the deterministic part of the data. This should give a clue to the region of data points that the cost function must converge on.

3.1.1 A simple problem

A Neural Network cost function must be devised to enable a network to learn to extract a deterministic pattern from a pattern that contains a deterministic part and a non-deterministic random part. It should be possible to learn the predictable behaviour and be satisfied at predicting the random part some of the time according to a probability distribution. To demonstrate the problem suppose a completely deterministic sequence of binary digits is presented consisting of '0, 1' pairs [1].

```
0 1 0 1 0 1 0 1 0 1 0 1
```

A Neural Network is required to predict the series of outputs when the input is subject to noise. The network will not know the pattern of the incoming data and it does not know that there is a function producing the noise. By using a cost function

[1]This problem was originally proposed by Nick Chater [Computer Science Department, Edinburgh University, personal communication.]

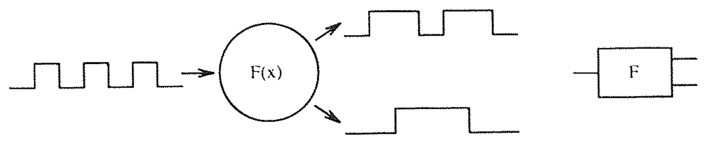

Figure 3.2: Adding noise to a deterministic pattern.

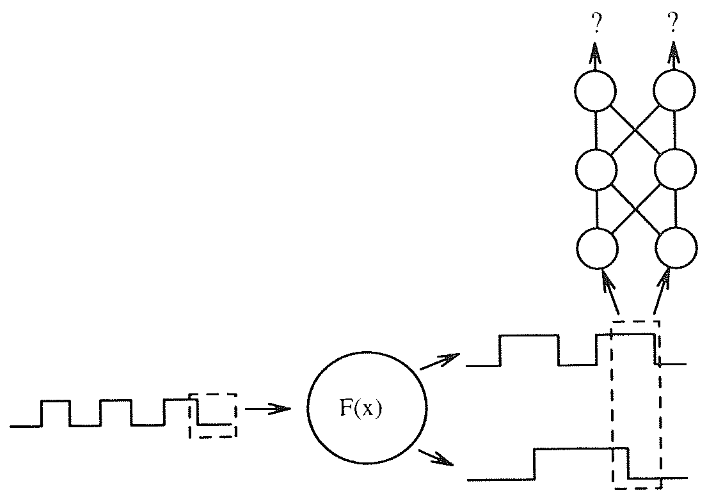

Figure 3.3: A network predicting the next two outputs from $F(x)$.

that converges to a mode it may be possible to extract some of the structure from the incoming data.

The sequence is the input to a stochastic function that simulates the addition of noise (adding the random part to the deterministic data) shown in Figure 3.2. The input is a single binary digit and the outputs are two binary digits. A single stream of binary digits will produce two streams of binary digits. The function $F(x)$ is,

$$F(x) = \begin{cases} (0,0) \text{ or } (1,1) & \text{if } x = 0 \quad \text{where } p(0,0) = 0.5 : p(1,1) = 0.5 \\ (1,0) \text{ or } (0,1) & \text{if } x = 1 \quad \text{where } p(1,0) = 0.5 : p(0,1) = 0.5 \end{cases} . \tag{3.1}$$

For example if the input is the binary digit '0' then the output will be (0,0) or (1,1), where the probability of each is 0.5. Similarly for the binary digit '1' the output could be (1,0) or (0,1) with equal probability. This function can be described as a probabilistic version of an XOR function or 'inverse' XOR. Instead of the XOR function being used to produce one output from two inputs, it is used to produce two outputs from one input.

Thus a single input to the function leads to two outputs. The outputs from the noise function are fed into a network to try to predict the next two outputs as shown in Figure 3.3. A multi-layer-perceptron (MLP) using the method of sum-of-squares cost function training may be used to predict the outcome for the next input. It may for example predict that the output (0,1) follows (0,0). However, it is known that an MLP using the sum-of-squares cost function will converge to a solution of 0.5, 0.5, the mean value.

An MLP using the sum-of-squares cost function will not be able to predict the next input. If it were true that the sequence of events were entirely deterministic then the Neural Network could accurately predict what would happen at the next time-step from the current time-step. The network could then be used to predict very accurately what would happen for many time-steps into the future. In Figure 3.3 only 1 set of inputs is used in the network inputs. In some time series prediction problems it may be useful to know about more than one time-step in the past.

The inputs in this problem have structure and the function producing noise is an XOR function. However the sequence of events has a random nature (the output could be one of two output patterns, which are entirely random). A cost function is required that will predict the deterministic part accurately and predict the random part correctly some of the time depending on the probability of that random part. It is preferable to be right half the time than be half right all the time. This is equivalent to learning a mode in a time series that contains a multi-valued mapping.

Another example not related to the extraction of a deterministic signal from noise is the control of robot arms that deals with multi-valued mappings. If a robot arm has two links there will be two ways of reaching the same point within the space of positions it must reach. For the robot arm to successfully grab the object its elbow must bend either up or down. As the number of links increases the number of alternative link angles that can be used to reach the desired position will increase. A Neural Network must converge on a particular set of link angles rather than converge on the mean solution where the links do not bend at all. In this case the mean will be a non-solution.

3.2 The conventional approach

A possible solution to solving multi-valued problems is to investigate the number of points fit by a model within a certain tolerance. A cost function is also required that will favour convergence on the largest cluster of data, or one of several multi-valued outputs. Before this is described in detail a typical cost function that would be used by a conventional Neural Network is described. This cost function gives the error over the data. The stiffness of the model is then described and how it can be used with the error over the data.

$$E_{Data} = \sum_p E_p \tag{3.2}$$

Cost functions typically take the form of a sum over all data points shown in (3.2). In conventional MLP's the Widrow-Hoff method is commonly used [92], also referred to as the delta rule. It is based on finding the difference between the desired output t and the actual output y on node i in pattern p,

$$e_{ip} = y_{ip} - t_{ip} \, . \tag{3.3}$$

The error for each node i in pattern p is a fixed function E of e_{ip}, $E_{ip} = E(e_{ip})$. In conventional MLP's the Widrow-Hoff method is based on the sum-of-squares cost function,

$$E_{Data} = \sum_{ip} (e_{ip})^2 \, . \tag{3.4}$$

Stiff models can be described as ones that have a small parameter space. Non-stiff models have a larger parameter space. As the parameter space is reduced the network is forced to fit the mean of a set of points if the sum-of-squares cost function is used. The stiffness can be controlled by a regularising parameter α. A common regularising technique is to penalise large weights, reducing the possible weight space where the solution can exist. The more weights are penalised the stiffer the model will be. Other researchers (LeCun *et al.* [41]) use second derivative information on the network solution to gain a smooth curve that fits the data. This penalty for large weights takes the form of another error term, which can be added to (3.4),

$$E_{Total} = E_{Data} + \alpha E_{Weights} \, . \tag{3.5}$$

The regularisation can be performed by a simple weight-decay term between all nodes in layer i and layer j,

$$\alpha E_{Weights} = \alpha \sum_{ij} w_{ij}^2 \, . \tag{3.6}$$

Here α is the regularising parameter used to penalise large weights. If α is 0 then the network is allowed to fit all the data. If α has a medium value then the curve will fit some points and not others. If α is infinite then the network mapping will be a straight line through the data. A new form for E is developed and it will be shown how regularisation can be used to solve multi-valued problems and show structure in the data.

3.3 The Naive Description Length cost function

There are two types of learning for which the new cost function may be used. These are supervised learning and unsupervised learning. Consider a data set in which the p^{th} pattern is the output $\mathbf{t}_p$. In supervised learning a set of inputs and a set of responses or target outputs are provided. A Neural Network learning to associate a pattern $\mathbf{x}$ with a pattern $\mathbf{t}$ is just a function mapping the input elements of $\mathbf{x}$ to the target output elements of $\mathbf{t}$ using a set of weights $\mathbf{w}$. It can also be described as a model F providing an estimate $\mathbf{y}_p = F(\mathbf{x}_p)$ of each 'target output' point $\mathbf{t}_p$, based on the 'input' point $\mathbf{x}_p$. Given the inputs, the outputs can be reconstructed using the function. The difference between the desired target output $\mathbf{t}_p$ and the network output $F(\mathbf{x}_p)$ gives the error.

In unsupervised learning a network must identify common features or clusters in the data without a set of desired responses. Input patterns can then be classified using these features or clusters. A crude model of a cluster is simply a point $\mathbf{y}$ that is near to several of the data patterns.

An unsupervised clustering problem can be regarded as a trivial supervised learning problem in which all inputs are identical or where there are no inputs. This is because supervised learning must estimate the probability distribution that models what the output will be for a given input. Unsupervised learning estimates the probability of the target data without any regard for the inputs. Therefore a supervised learning problem can be regarded as a set of related unsupervised clustering problems. If there can be two or more different output values for the same input value, then a separate unsupervised clustering problem exists for each input value.

The network must converge on a particular cluster if there are multiple output values for the same input value. A cost function must reward a model that has a good approximation of the points within a cluster more than it penalises it for a poor approximation of points far away from the cluster. The cost function should favour a good fit to a few points rather than a mean fit to many points.

This idea can be expressed by considering the accuracy required to model the data at the scale of a cluster size. We may desire that the accuracy of the network model be within a certain limit ϵ. This means that elements that give an error greater than the level of accuracy should not form part of the model,

$$| F_{ip} - t_{ip} | < \epsilon \, . \tag{3.7}$$

For the new cost function, again the difference between the desired output t minus the actual output y on node i in pattern p is calculated. We require that the error over the data $e_{ip} = y_{ip} - t_{ip}$ should increase relatively rapidly with $| e_{ip} |$ for $| e_{ip} |$ on the scale of the cluster size, but relatively slowly at other scales.

3.3.1 Description length

One interesting cost function that embodies these features is an expression for the cost of encoding a set of data by a naive method (Rohwer and van der Rest [71]). A data set consists of a number of patterns p giving an output t_p. The data is to be encoded within accuracy ϵ. Each output can be expressed with an estimate y_{ip} as $t_{ip} = y_{ip} - e_{ip}$. Given the network output and the error the original output can be reconstructed. The 'error' can be measured within a certain accuracy ϵ in the same way as the target t_p. The function $\lceil x \rceil$ is used, which gives the nearest integer greater than or equal to x, the error $e_{ip} \approx \lceil \frac{|e_{ip}|}{\epsilon} \rceil \epsilon$. The error e_{ip} can be represented by an integer $\lceil \frac{|e_{ip}|}{\epsilon} \rceil$ at a cost of about $\log_2 \frac{e_{ip}}{\epsilon}$ bits, plus a sign bit. For example if $\mid e_{ip} \mid= 0.519$ and $\epsilon = 0.01$ then the cost would be $\log_2(52)$ bits ($\lceil 51.9 \rceil = 52$).

$$E_{ip} = \log_2 \left(\frac{\mid e_{ip} \mid}{\epsilon} \right) \tag{3.8}$$

More information may be introduced to separate numbers that have different lengths. Encoding different length numbers could be done by preceding each with a number encoding the bits required, $\log_2(\log_2(\max_p(e_p)))$. This gives the number of bits encoding the largest possible description of the error. The aim of using this cost function is to provide an interpretation of the minimum description length. A method that eliminates the need for separators is Huffman coding, which could be used if data compression was the main objective. This method also provides a more optimal assignment of error values to integers. For the cost function described here the fixed per-number costs of sign bits and separator fields, and the fixed costs of encoding ϵ and generating $\mathbf{y}_p$ are ignored. This is why the cost function is called the Naive Description Length cost function.

The cost of encoding the data E_{Data} is as follows:

$$\begin{aligned} E_{Data} &= \sum_{ip} \max \left(\log_2 \frac{\mid e_{ip} \mid}{\epsilon}, 0 \right) \\ &= \sum_{ip} \log_2 \left(\max \left(\frac{\mid e_{ip} \mid}{\epsilon}, 1 \right) \right) . \end{aligned} \tag{3.9}$$

This shows that, for larger values of ϵ, smaller values of information are required. The equation makes sure that only logarithms of positive numbers are taken. It also makes sure that errors outside the scale of ϵ have no significance. The value of ϵ provides a lower bound on the amount of error, so the greater of the two must be used. This applies to each component of the error vector for each example.

We require ϵ as a scale over entire error vectors. For vectors of outputs the following adjustment is required:

$$E_{Data} = \sum_p \log_2 \left(\max \left(\frac{\sum_i \mid e_{ip} \mid}{\epsilon}, 1 \right) \right) \tag{3.10}$$

$$= \sum_p \log_2 \left(\frac{\max(\sum_i \mid e_{ip} \mid, \epsilon)}{\epsilon} \right) . \tag{3.11}$$

Equation 3.11 shows that the maximum of the error and ϵ is taken. This can be represented as the $n \to \infty$ limit of the following differentiable cost function:

$$E_{Data} = \sum_p \ln \left[\left(\frac{\sum_i \mid e_{ip} \mid^n + \epsilon^n}{\epsilon^n} \right)^{\frac{1}{n}} \right] . \tag{3.12}$$

Changing from base 2 logarithms to natural logarithms introduces a constant factor, which can be ignored. The case of $n = 2$ is considered in this thesis,

$$E_{Data} = \sum_p \ln \left(\frac{\sum_i \mid e_{ip} \mid^2 + \epsilon^2}{\epsilon^2} \right) . \tag{3.13}$$

The accuracy can be represented by the amount of information required to achieve the mapping from the inputs to the outputs. When the accuracy required of a model is low the value of ϵ must be large and therefore the information required to reconstruct the output will be low. When the accuracy required of a model is high the value of ϵ must be small and so the information required to reconstruct the output will be high.

3.3.2 Minimum descriptions and clustering

The new cost function will identify clusters in the data by fitting as many points as possible within a given scale. The production of minimum descriptions leads to models that express clusterining information. This is because an efficient code must make use of the structure of the data distribution. It must assign the shortest codes to points in the most populated and compact clusters, as determined by one means or another (Rohwer and van der Rest [71]).

3.3.3 The noise model for the NDL cost function

Assuming that the probability of the targets t given y is equivalent to an exponential of the cost function:

$$P^{(NDL)}_{n\epsilon\beta}(t \mid y) = \frac{1}{Z^{(NDL)}} e^{-\beta E^{(NDL)}(y,t)} = \frac{1}{Z^{(NDL)}} \left(\frac{\epsilon^n}{\mid y - t \mid^n + \epsilon^n} \right)^{\frac{\beta}{n}} , \tag{3.14}$$

$$P^{(NDL)}_{n\epsilon\beta}(D) = \frac{1}{Z^{(NDL)}} e^{-\beta E^{(NDL)}(Y,T)} = \frac{1}{Z^{(NDL)}} \prod_p \left(\frac{\epsilon^n}{\mid y_p - t_p \mid^n + \epsilon^n} \right)^{\frac{\beta}{n}} . \tag{3.15}$$

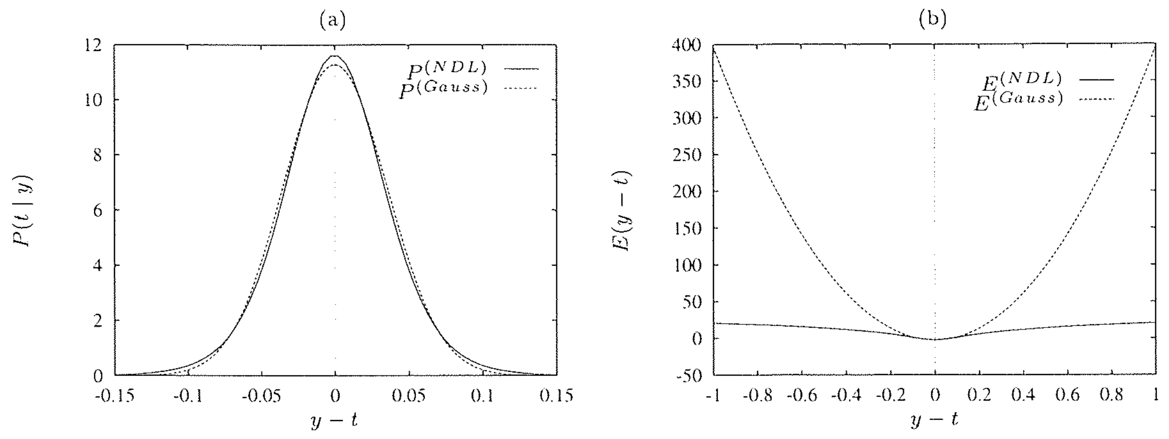

Figure 3.4: Comparing the noise model for the NDL and sum-of-squares cost function. (a) $P^{(NDL)}$ for $\epsilon = 0.1$, $n = 2$, and $\beta = 10$, together with $P^{(Gauss)}$ for $\epsilon/\beta = 0.05$ using different values for ϵ and β. The central peaks are similar, but the tails decay faster in the Gaussian. (b) Error measures for noise models in (a) (simply their negated logarithms). Note the different horizontal scale.

Input	Output
1	0
1	1

Table 3.1: Training set for a simple 2 pattern multi-valued problem.

Figure 3.4 shows the noise model for both the sum-of-squares and NDL cost function. They show the likelihood of the target t given the network output. The two models look very similar, but close inspection of the tails show that the NDL noise model gives a higher probability to outliers. This is because the NDL cost function is insensitive to these outliers. Figure 3.4b shows that errors are regarded in a similar way.

3.3.4 A simple multi-valued mapping problem

The following problem shows in the simplest terms how such a cost function would work. This problem contains a multi-valued mapping where the single input '1' maps to '0' and '1'. The training set is shown in Table 3.1.

A conventional MLP trained on these two input-output pairs will be expected to converge on the mean output of 0.5. The conventional sum-of-squares error between each target pattern and the range of network outputs can be calculated. The output from the network will fall in the range 0 to 1. For example if the output was 0.2 then the error defined as the output minus the target would be 0.2 and 0.8 respectively for

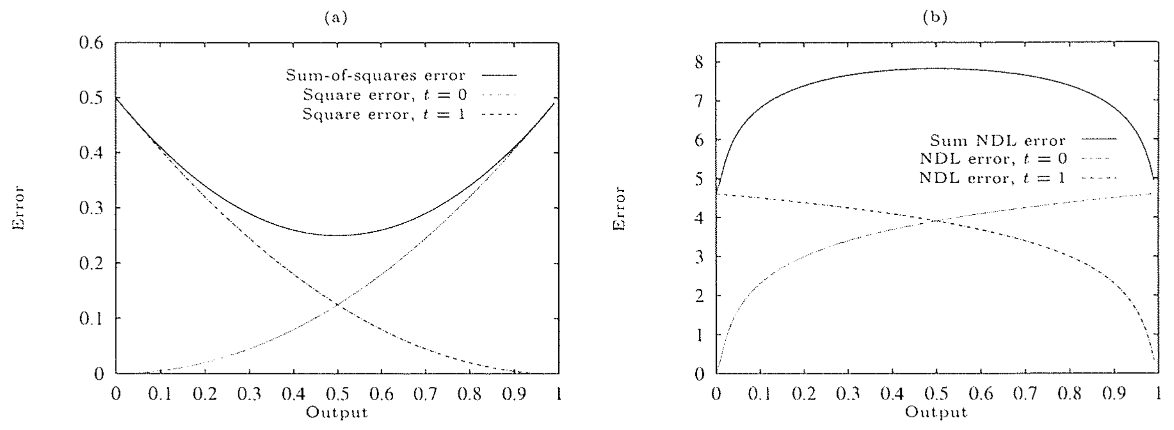

Figure 3.5: The error produced for individual patterns and their sums. (a) The error produced when using the sum-of-squares cost function. (b) The error produced when using the Naive Description Length or NDL cost function with $\epsilon = 0.01$.

each pattern. Figure 3.5a shows a minimum error when the output is 0.5.

When the error is calculated for each output using the Naive Description Length cost function then the error will look like that shown in Figure 3.5b. Here the value of ϵ is 0.1 and so two minima occur. One minimum is where the output is nearly '0' and another where it is nearly '1'. The total error when the output is '0' or '1' is less than when it is '0.5'. If the error is high relative to the value of ϵ then the increase in cost that it incurs will always be less than the decrease in cost of trying to fit a point within the range of ϵ where the error is relatively small. If a point is within ϵ of the current solution then the solution will be attracted to this point. Therefore a small ϵ value prefers some points over others. For example if the output is 0.2 in Figure 3.5b the error for the pattern '1' to '0' is decreasing faster than the increase in error for the '1' to '1' pattern.

As the value of ϵ is increased the level of accuracy and hence information decreases and the error produced for each output reverts to one similar to the sum-of-squares error where the minimum error is '0.5'. This can be seen in Figure 3.6, which shows how the total error varies for different values of ϵ, again showing the total of the individual pattern errors. As ϵ increases the differences in error for the individual patterns have a lesser impact on which solution is chosen and the network solution moves more to the mean of the two. The value of ϵ determines on which scale two or more points need to be distinguished. When the value of ϵ becomes greater than that needed to distinguish between two points the cost function reverts to finding the mean.

All this conveniently ties in with information theory, and Equation 3.12 represents approximately the minimum information needed to reconstruct the data. Equation 3.12 increases most rapidly on scales of about ϵ. Its logarithmic increase at large scales

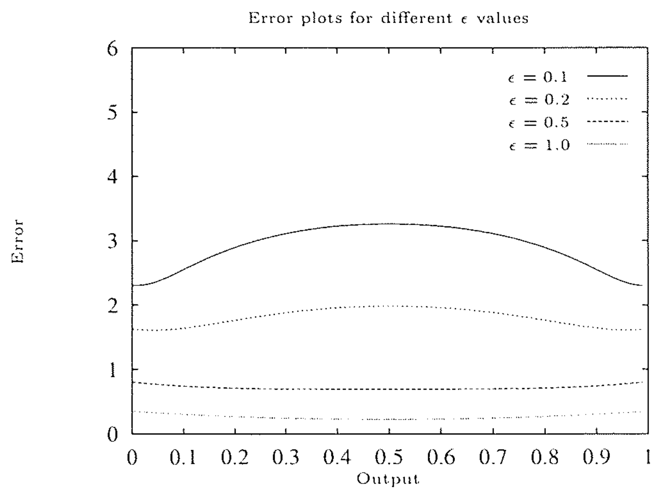

Figure 3.6: How the Naive Description Length error is similar to the sum-of-squares error for high values of ϵ.

lightens the impact of points that fit very poorly. Therefore, considering the discussion above, this cost function should guide models to clusters roughly of size ϵ. More examples will demonstrate that this is the case.

3.3.5 Regularisation and the number of 'well-fit' points

Different values of α and ϵ can be used to test how they affect the behaviour of the network. The constant ϵ defines the exactness of fit tolerance within which points should lie. A linear transformation of Equation 3.13 can be used to obtain a measure of the points that are 'fit well' by the network:

$$\begin{aligned} E_{Data} &= \sum_p \ln\left(\sum_i \mid e_{ip}^2 + \epsilon^2\right) - \ln \epsilon^2 \\ E_{Data} + \ln \epsilon^2 &= \sum_p \ln\left(\sum_i \mid e_{ip}^2 + \epsilon^2\right) \\ \frac{E_{Data} + \ln \epsilon^2}{\ln \epsilon} &= \frac{\sum_p \ln\left(\sum_i \mid e_{ip}^2 + \epsilon^2\right)}{\ln \epsilon} \, . \end{aligned} \tag{3.16}$$

As $\epsilon \to 0$ the NDL cost function can be used to obtain a measure of how many points are fit well by the model,

$$N_{2,\epsilon}^{(NDL)} = \sum_p \frac{\ln\left(\sum_i \mid e_{ip} \mid^2 + \epsilon^2\right)}{\ln \epsilon} \, . \tag{3.17}$$

The more general measure for different n is,

$$N_{n,\epsilon}^{(NDL)} = \sum_p \frac{\ln\left(\sum_i \mid e_{ip} \mid^n + \epsilon^n\right)^{\frac{1}{n}}}{\ln \epsilon} \, . \tag{3.18}$$

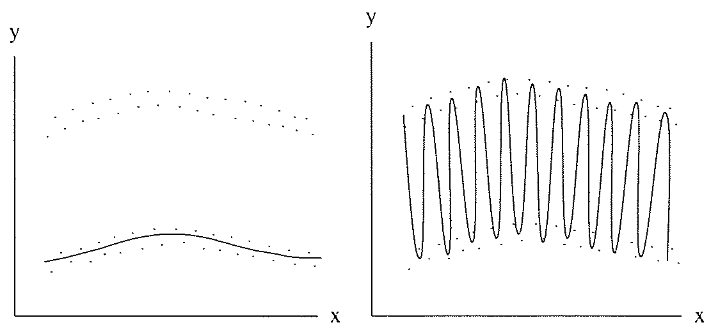

Figure 3.7: The 'snap' effect as the regularising parameter α increases.

If the error for a particular pattern is zero then the numerator in Equation 3.17 is $\ln \epsilon$ making the 'cost' for that pattern 1. This is a well-fit point. If the error for a particular pattern is not zero then the numerator will not be $\ln \epsilon$ making the 'cost' approach zero for any large error.

The regularising parameter α controls the stiffness of the model. It is used to control how much the weights are penalised. Graphs can be generated showing the dependence of the error to the amount of regularisation. It is predicted for many problems, such as regression problems, that as the stiffness of the model is decreased there will be a 'cliff' in the graph. A small change in the stiffness will lead to a sudden increase in the number of points fitting within the tolerance. The solution will appear to 'snap' from fitting only some of the data in a particular region to fitting much more of the data. This should happen in cases where the data has some structure. The Naive Description Length cost function is capable of showing multiple levels of structure in the data. Figure 3.7 shows an example problem with two clusters of data. This type of problem is examined in more detail in Section 3.4.5.

When the stiffness is high the network will try to follow one set of points rather than the other. As α decreases and the regularising parameter α approaches 0 the network will try to fit all points within the tolerance as shown in Figure 3.7.

It is predicted that the points of data the network will choose to follow will depend on the number of data points fitted. For example if a data set contains two clusters describing parabolas, but have a different number of points, the parabola described by the most points will be chosen. For high values of α the network will fit clusters of data where there are more points. Again a 'snapping' effect will occur when the network fitting a cluster of points will be flexible enough to prefer to fit all the data. This change will occur over a small range of α because of the absence of any structure at any other scale. This may give clues to the appropriate amount of regularisation to achieve good results for test sets.

3.4 Testing the cost function on some toy problems

The cost function was first tested on an unsupervised problem. A supervised problem was then used to show the effect ϵ has on training. The effect of regularisation and the number of well-fit points was investigated for both types of training. Finally the application of the inverse kinematics of Robot arms is discussed.

3.4.1 Unsupervised learning

The new cost function was first tested on an unsupervised learning problem. In conventional unsupervised learning a network will be able to identify clusters in the input. The aim of using the NDL cost function in unsupervised learning was to see if different values of ϵ used in training would identify clusters in the data at different scales.

In this test an unsupervised learning network was used. The training data was chosen from a probability distribution consisting of a mixture of three Gaussian probability distributions. The centres were 0.25, 0.65 and 0.85. The corresponding variances σ were 0.06, 0.02 and 0.04 respectively. Each point was found by finding the limit g of the integral of the sum of three Gaussian functions, which would give an area equal to a random number selected from a uniform distribution between 0.0 and 1.0.

$$r = \frac{1}{3}\sum_{i=1}^{3}\int_{-\infty}^{g}\frac{1}{\sqrt{2\pi}\,\sigma_i}e^{\frac{-(x-c_i)^2}{\sqrt{2}\sigma_i^2}}\,dx\,, \tag{3.19}$$

where c_i and σ_i refer to the centres and variances. The area of the integral must be equal to the random number r. Integrating equation (3.19) and using the function $erf(x)$ an iterative binary search can be performed to find a value of the limit g,

$$r = \frac{1}{6}\sum_{i=1}^{3}\left[1 + erf\left(\frac{g-c_i}{\sqrt{2}\,\sigma_i}\right)\right]\,. \tag{3.20}$$

Details of this procedure can be found in Appendix A.7.

The training set contained 1000 points and the network was trained with a range of values for ϵ. Figure 3.8 shows how the number of points that fit within the scale of ϵ increases and affects the output of the network. When ϵ is 0.05 the output is 0.66. This is very close to the Gaussian probability distribution centred at 0.65. This distribution also has the smallest variance of 0.02. The random data is spread over a smaller range so the number of points fitted at the scale of ϵ for this distribution is greater than the others. As ϵ increases, points from the distribution centred at 0.85 fall within the scale of ϵ. The cost function will cause the network to converge to an output that is the mean of all the points within the scale of ϵ. When ϵ reaches a value of 0.9 all the points fall inside the scale of ϵ and the output draws closer to the mean of the entire training set, which is 0.59.

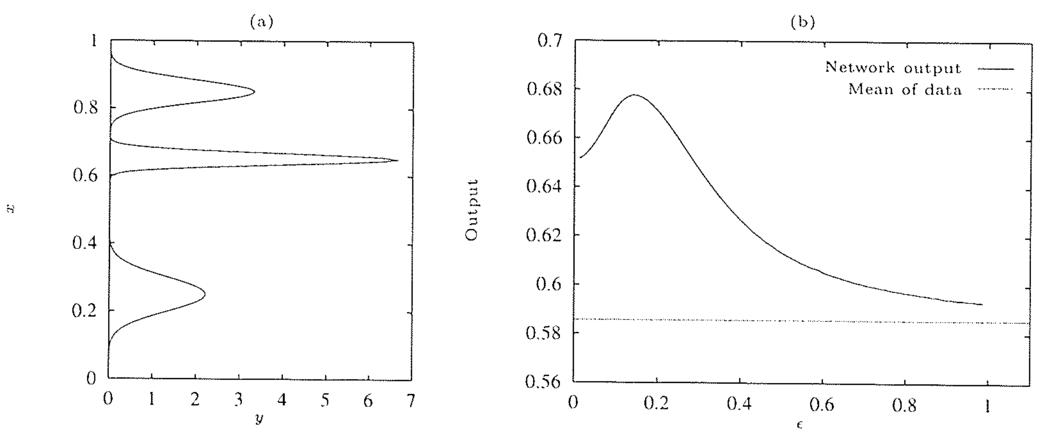

Figure 3.8: As ϵ increases the number of points that fit within the scale of ϵ increases. (a) The Gaussian probability distribution between 0 and 1. (b) The output of the network between 0.56 and 0.7 as the value of ϵ is varied.

The number of well-fit points can be calculated for each value of ϵ as shown in Figure 3.9. The number of well-fit points increases as more points fall within the ϵ scale and are regarded as 'close-fitting'. As the value of ϵ approaches 1 the interpretation breaks down (Rohwer and van der Rest [71]).

3.4.2 How ϵ affects the network output

The last section showed how the NDL cost function would perform for an unsupervised problem, where the network was required to model a distribution of a single output. In supervised learning the distribution of outputs is input dependent. In Section 3.1.1 the problem of a deterministic pattern subject to random noise was described. A probabilistic 'inverse' XOR function was used to create the noise, see Equation 3.1. This problem shows that there is a deterministic part to the data produced from the noise function in that a different type of noise occurs for each input in a particular sequence of inputs. If the sequence of inputs is deterministic then the type of noise that occurs must also be deterministic. There is also a random part, in that the noise within each type is entirely random. The aim of the experiment is to look at the pattern of noise that occurs as a sequence and to try to predict the next set of outputs from the noise function. A Neural Network is required to predict which type of noise will occur next. The network will not however be able to predict accurately the actual noise that will occur.

For this experiment the training set shown in Table 3.2 was used. The training set inputs show all the different combinations that the noise function can produce. The noise function was applied to a repeating pattern of '0 1's. The output from the noise

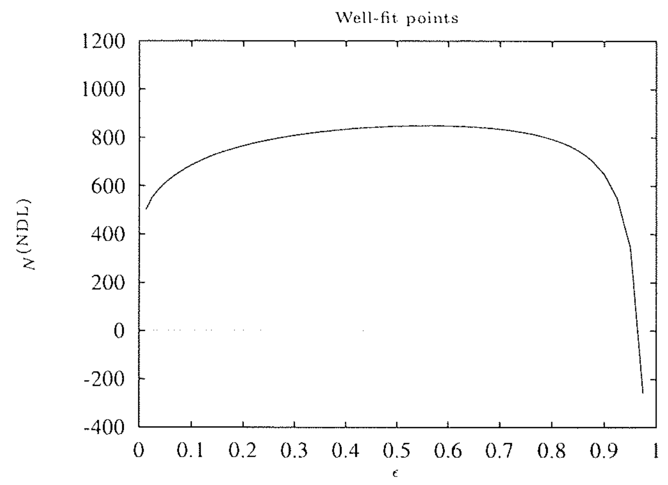

Figure 3.9: The number of 'well-fit' points for varying values of ϵ.

Inputs	**Target Outputs**	$\epsilon = 0.05$ **Outputs**	$\epsilon = 0.4$ **Outputs**
0.0 0.0	0.0 1.0	0.0 1.0	0.2 0.8
0.0 1.0	1.0 1.0	0.0 0.0	0.8 0.8
1.0 1.0	1.0 0.0	0.0 1.0	0.8 0.2
1.0 0.0	0.0 0.0	1.0 1.0	0.2 0.2
0.0 0.0	1.0 0.0	0.0 1.0	0.2 0.8
1.0 0.0	1.0 1.0	1.0 1.0	0.2 0.2
1.0 1.0	0.0 1.0	0.0 1.0	0.8 0.2
0.0 1.0	0.0 0.0	0.0 0.0	0.8 0.8

Table 3.2: The training set for the signal and noise problem is shown in the first two columns. Training with $\epsilon = 0.05$ and with $\epsilon = 0.4$ are shown.

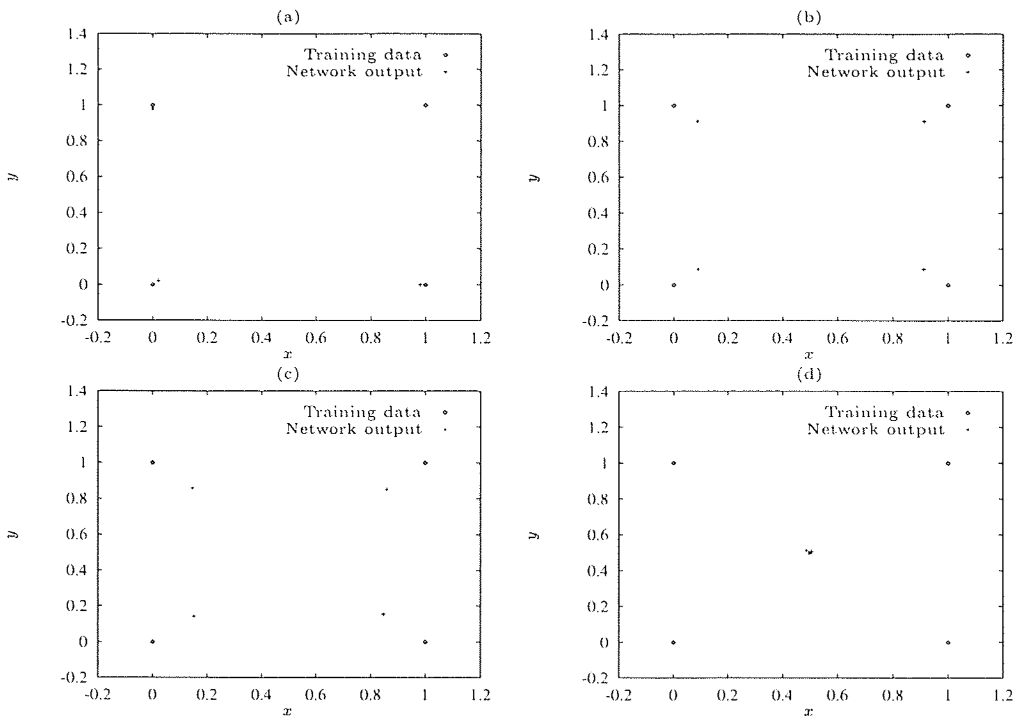

Figure 3.10: When the value of ϵ is small the outputs closely match the possible inputs in an 'inverse' XOR training set. As ϵ increases the outputs start to converge to the mean. The outputs converge to the mean when ϵ is large. (a) $\epsilon = 0.2$. (b) $\epsilon = 0.4$, (c) $\epsilon = 0.5$. (d) $\epsilon = 0.8$.

function formed the inputs to the network. The target output data for each pair of inputs in Table 3.2 is the next pair of outputs produced by the noise function. A Neural Network was required to learn the rule that (0,0) or (1,1) always follows (0,1) or (1,0) and that (0,1) or (1,0) always follows (0,0) or (1,1). As shown in Equation 3.1 the network must converge on either of the outputs (0, 0), (0, 1), (1, 0) or (1, 1).

The outputs in Table 3.2 show the predicted outputs from a network using the Naive Description Length cost function. A standard gradient descent backpropagation network was used with a single hidden layer with 4 nodes. The network was trained with ϵ set to 0.05 and was made to choose combinations very close to the possible combinations. The network could not predict the non-deterministic part of the inputs but successfully learned the deterministic part. In Table 3.2 the first prediction that the next input would be (0,1) is correct. The next prediction is wrong, predicting a (0,0) combination when (1,1) was the input. As far as predicting the non-deterministic part, the network has a probability of 0.5 in getting the prediction correct.

Other experiments were carried out with different values of ϵ. When the value of ϵ is small the network outputs closely match the possible combinations of the noise function, as shown in Figure 3.10a when ϵ was set to 0.2. The value of ϵ is at a scale small enough to distinguish between all the possible outputs of the noise function. As ϵ increases the outputs start to move toward the mean, which is 0.5, 0.5. This would be the solution a conventional MLP would converge to. When ϵ is 0.4 the outputs clearly move towards the mean as shown in Figure 3.10b. When the value of ϵ is 0.8 the outputs are close to the mean as shown in Figure 3.10d.

The results for different values of ϵ only show the possible outputs of the network to each of the inputs. That the network must alternate between [(0,0) or (1,1)] and [(0,1) or (1,0)] is not shown by the presentation of the results.

3.4.3 Inverse Kinematics

To test the Naive Description Length cost function on a simple application the Inverse Kinematics problem was chosen. This is an application where the most probable output is of greater interest than the average output. The cost function was tested on the problem outlined by Bishop [6] that consisted of a two link Robot arm. The end effector is moved to a particular position by setting the correct angle for each link. The position of the end effector with a particular set of angles is calculated by the following two equations,

$$x_1 = L_1 cos(\theta_1) - L_2 cos(\theta_1 + \theta_2)\ , \tag{3.21}$$

$$x_2 = L_1 sin(\theta_1) - L_2 sin(\theta_1 + \theta_2)\ , \tag{3.22}$$

where L_1 and L_2 are the lengths of the two links. The forward kinematics of the robot arm maps the angles (θ_1, θ_2) to a position (x_1, x_2) and is a single-valued mapping. The inverse kinematics of the robot arm maps the position (x_1, x_2) to the angles. It turns out that for certain positions the set of angles describing that position is multi-valued. Two orientations of the robot links give the same position. A network trained using the NDL cost function should converge on one branch of the mapping from the Cartesian space to the joint angle space. In Bishop [6] a particular problem was outlined where the angles θ_1 and θ_2 were restricted to the ranges $(0.3, 1.2)$ and $(\frac{\pi}{2}, \frac{3\pi}{2})$ respectively. The lengths of the links were $L_1 = 0.8$ and $L_2 = 0.2$. Figure 3.12 shows the details of the links.

A training set was created by randomly generating the two angles of the links restricted to the ranges above. The equations 3.21 and 3.22 were then used to calculate the end effector positions. A test set was generated in a similar way using different random angles. Each set had 1000 random angles and positions in it. A large number of examples would then give a number of positions where the link orientations or angles would be different. The training set end effector positions are shown in Figure 3.11.

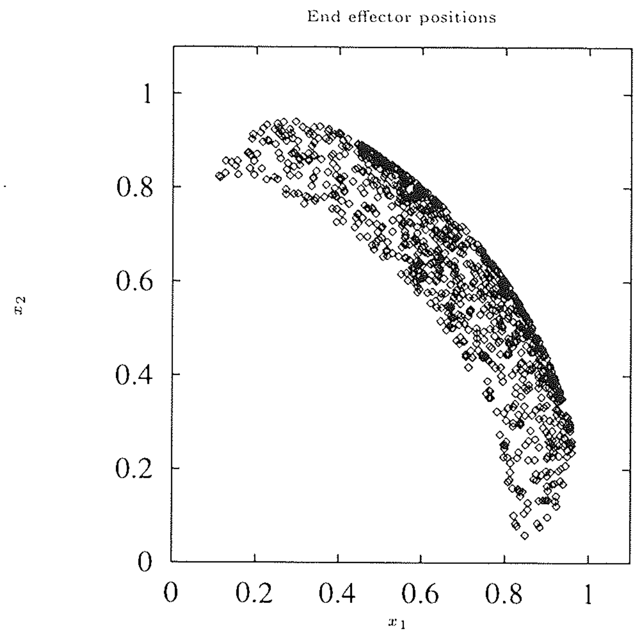

Figure 3.11: A training set of positions that a robot arm can reach.

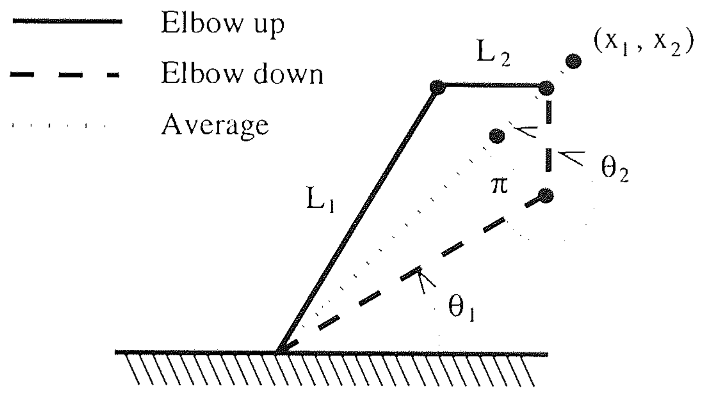

Figure 3.12: A Robot arm in the elbow up and elbow down position. It also shows the mean position where the second arm link is at an angle of π.

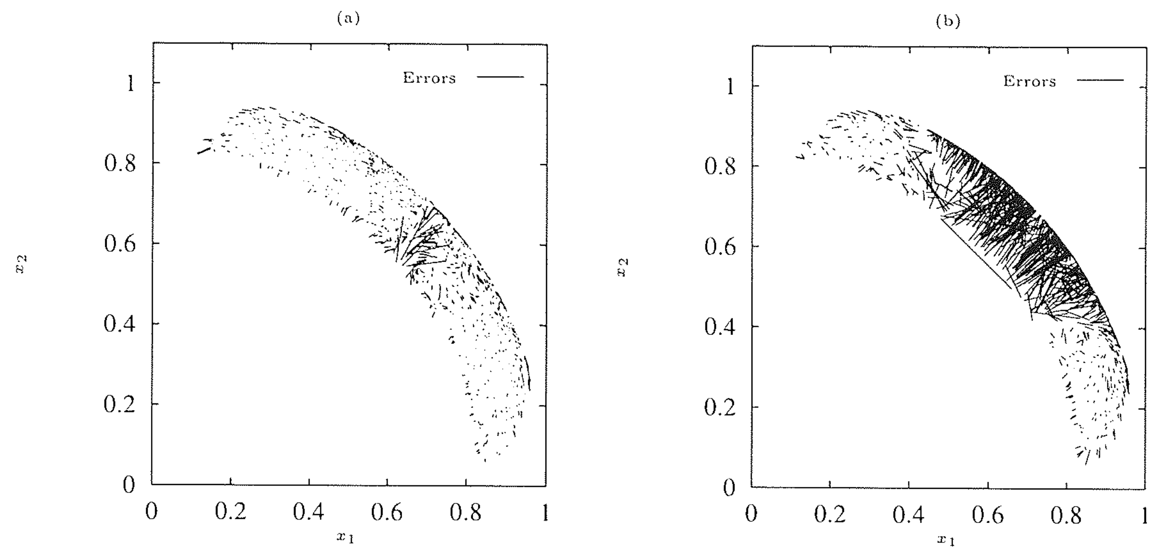

Figure 3.13: Robot position errors for different methods of training. (a) A network trained using the NDL cost function. (b) A network trained using the sum-of-squares cost function.

A standard gradient descent backpropagation network using the Naive Description Length cost function was used with a single layer of 100 hidden nodes and trained with 6000 iterations. The end effector positions using the angles predicted by the network were calculated and the RMS Euclidean distance found between the predicted and actual end effector positions. Figure 3.13a shows the errors produced after training using the NDL cost function. The robot arm can reach all parts of the space available. However, some large errors occur where the robot arm must flip from an elbow up to elbow down orientation. The RMS error for the training set was used to tune the parameters of the network such as ϵ and the regularising parameter α. The smallest RMS error of 0.00644 was obtained with an ϵ value of 0.045 and an α value of 10^{-6}.

The RMS error for the Naive Description Length cost function can be compared with the result obtained by Bishop [6] for the sum-of-squares cost function, which gave an RMS error of 0.0578. Bishop [6] shows that, by training a standard Neural Network by minimising the sum-of-squares cost function, the mean of the elbow up, elbow down will be found. This is shown in Figure 3.12b in regions where there is a multi-valued mapping. Bishop [6] goes on to show that Mixture Density Networks are capable of solving the multi-valued mapping.

The use of RBF networks for this problem gave even better results shown in Figure 3.14 and gave an RMS error of 0.000311. The number of basis functions used matched the number of patterns in the training set. Only a few large errors occurred where the robot arm must switch orientation to reach certain positions.

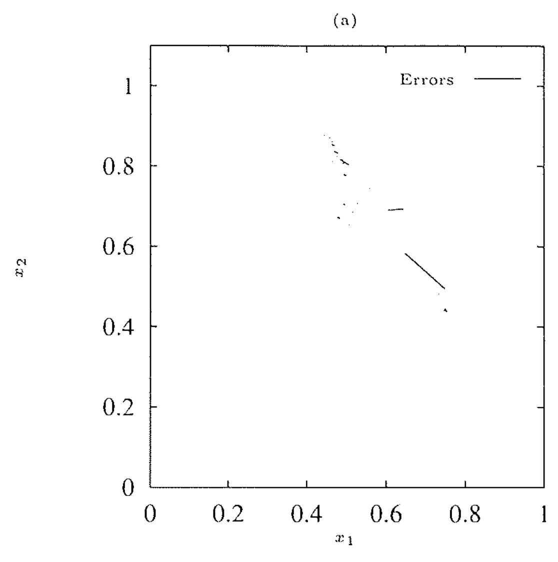

Figure 3.14: Robot position errors for an RBF network trained using the NDL cost function.

The sum-of-squares cost function can be used to approximate the conditional average of the targets given the inputs by minimizing the sum-of-squares error through training. The NDL cost function has proved useful in solving multi-valued problems. As we have seen the use of the sum-of-squares cost function has its limitations for multi-valued problems. Another method that gives a more detailed description of the probability distribution of the targets conditioned on the inputs is the Mixture Density Network already mentioned. This method combines a mixture density model with a conventional neural network. The probability distribution is a linear combination of a set of kernel functions. Bishop [6] considers kernels that are Gaussian. The parameters such as the mixing coefficients of the kernels, their means and variances are found by using a conventional Neural Network. For the Inverse Kinematices problem the RMS positioning error achieved was 0.0053. However, the selection of the correct model is more complex. Mixture Density Networks have the added problem of determining the correct number of kernel functions.

3.4.4 Problems with the NDL cost function

For the Inverse Kinematics of a robot arm problem it was noted that the network had difficulty in finding the right solution in certain areas. This is because of a need for the robot arm to suddenly flip from an elbow down to an elbow up orientation to reach particular positions.

Figure 3.15 shows all the possible end effector positions given the allowable angles

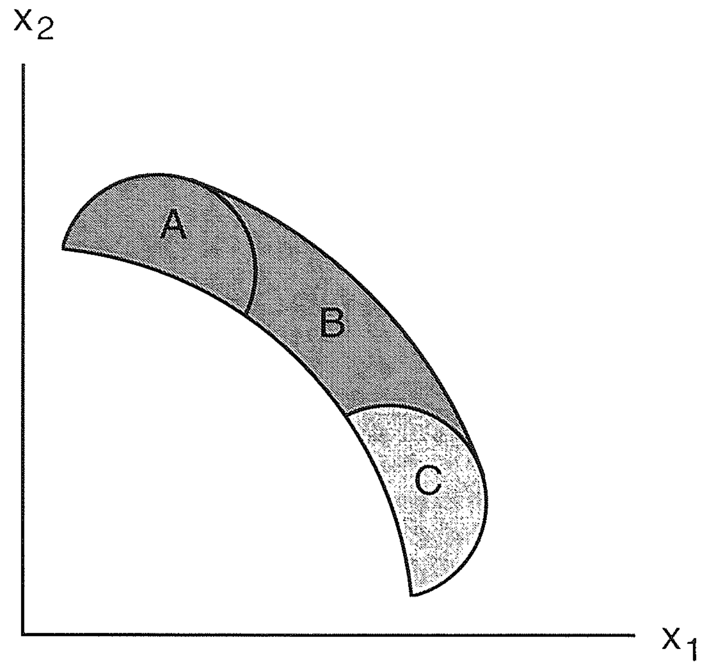

Figure 3.15: The two regions showing the different orientations of the Robot arm.

for each link arm. The regions A and B show the possible end effector positions reachable by an arm where the second link arm has an angle θ_2 between π and $\frac{3\pi}{2}$. Training has caused the network to choose angles where the arm bends in one particular direction. In this case the elbow is down throughout region B. Not all end effector positions can be reached with the elbow down and these positions are shown by region C where the second link arm has an angle θ_2 between $\frac{\pi}{2}$ and π. Along the boundary between the two regions lies an area where two positions very close to each other have their elbows up and down. The model is not flexible enough to provide the discontinuity between the two regions because Neural Networks model continuous functions.

This can also be seen where the cost function is used to interpolate the inverse of the following function (from Bishop [6]),

$$x = t + 0.3sin(2\pi t) + \varepsilon\ . \tag{3.23}$$

The variable t is the input and x is the output. Noise is added to the function by the term ε, which has a uniform distribution over the range (-0.1, 0.1). The value of t is restricted to the range 0.0 to 1.0. A training set was created that contained 1000 input-output pairs. To obtain the inverse problem the outputs x where used as inputs and the inputs t were used as the outputs.

A network trained using the NDL cost function converges on the solution in Figure 3.16 where the network has 100 hidden nodes and a regularising parameter α set to

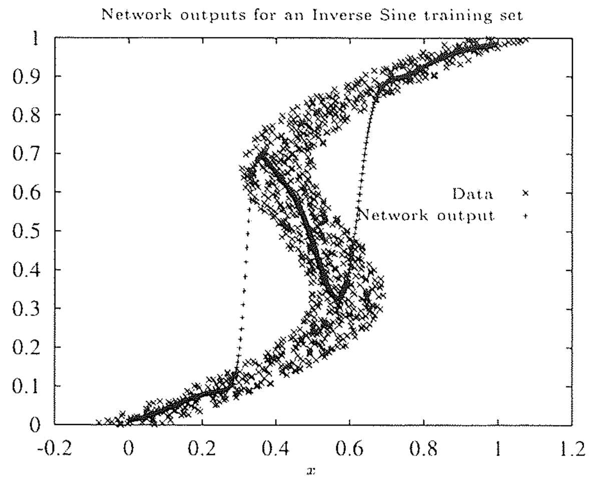

Figure 3.16: How the NDL cost function models the discontinuous regions in an inverse sine training set.

0.01. However there are some points that do not fit within the cluster of points. Again this is because the model is not flexible enough to provide the discontinuity in the data. A possible solution to this would be to identify the areas where large errors occur and split up the training data. Different networks can be trained for each segment of the training data.

3.4.5 How regularisation affects the network output

Regularisation affects network training by altering the way a network solution fits the data. The next experiment shows how regularisation affects training when the Naive Description Length cost function is used. A data set may be known to have interesting structure, *a priori*, at an unknown scale. This structure can be discovered by training with the NDL cost function and measuring the NDL data error for a range of different values for the regularising parameter α. This is called a training α-survey. Figure 3.17a shows 10 data points whose outputs have two values, 0.3 and 0.6. The data is not strictly multi-modal where multiple outputs exist for a single value of x. In this case each value of x has a unique output y. However the data set can be interpreted as having two clusters existing along two horizontal lines for the range of values for x. The aim of this experiment was to show that the cost function is capable of following either the bottom set of points or the top set. Another aim was that regularisation would show the presence of these clusters in the data. Again a standard gradient descent backpropagation network was used. The network was trained using a single hidden layer of 100 nodes for 1000 iterations to make sure all points were fitted for small values of α. The inputs were coarse-coded to allow the network to fit all

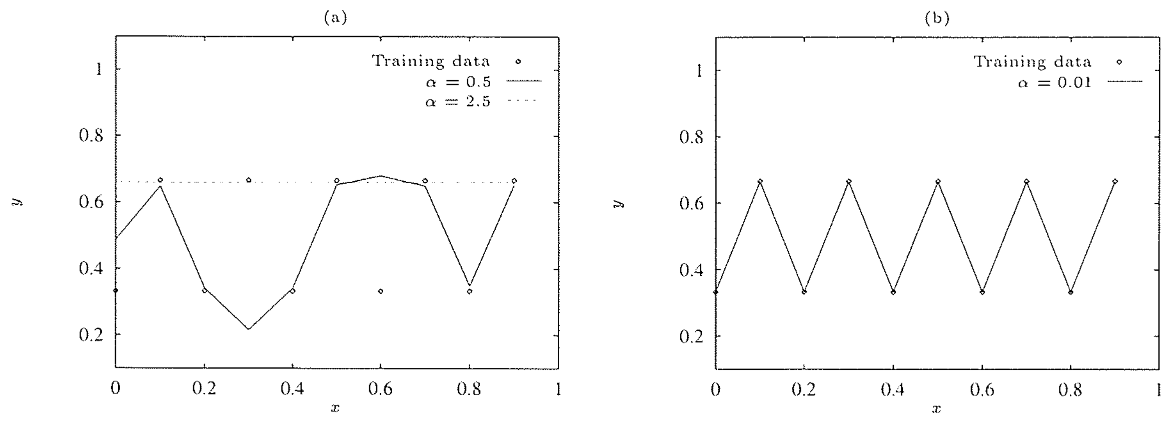

Figure 3.17: The two-lines training data set and network outputs for showing the effects of using different values of α in training. (a) For a small range of α the number of points fitted by the network model rises sharply. (b) The training data and the network outputs when $\alpha = 0.01$.

the data. Coarse-coding increases the dimensionality of the inputs and is explained in Appendix A.8. The value of ϵ was kept constant at 0.05 as the value of α was varied in an α-survey.

For these experiments the value of α was decreased from a large to a small value. For values of α of 0.75 and greater the network settles on one of the two sets of points, in Figure 3.17a this is the top one when $\alpha = 2.5$. At this point the network has found structure on the largest scale. A similar output of the network converging to a straight line through the points at 0.33 is equally probable. Over a small range of values for α from 0.5 down to 0.25 the network output changes rapidly. At this stage the output of the network is uncertain, as shown in Figure 3.17a. Figure 3.17b shows how the network has fit all the data when the value of α is 0.01. The network has trained on the structure that exists at the smallest scale. As the value of α decreases the network continues to fit all the data. No test set was generated for this problem, the interest being only in the behaviour of the training α-survey.

The training set NDL data error for each value of α was plotted and is shown in Figure 3.18a. As the value of α is decreased over a small range there is an obvious decrease in the NDL data error. This shows that the network solution suddenly snaps down to converge on many points instead of just a few. Two regions of the α survey show constant NDL data error. This suggests a value to set the regularising parameter α to give the different types of structure. Training with the sum-of-squares cost function appears to give the same sharp decrease in error as α is decreased in Figure 3.18b. However there is a rapid decrease to an error of zero for NDL cost function training. This effect will be seen more clearly for experiments with many more data points in

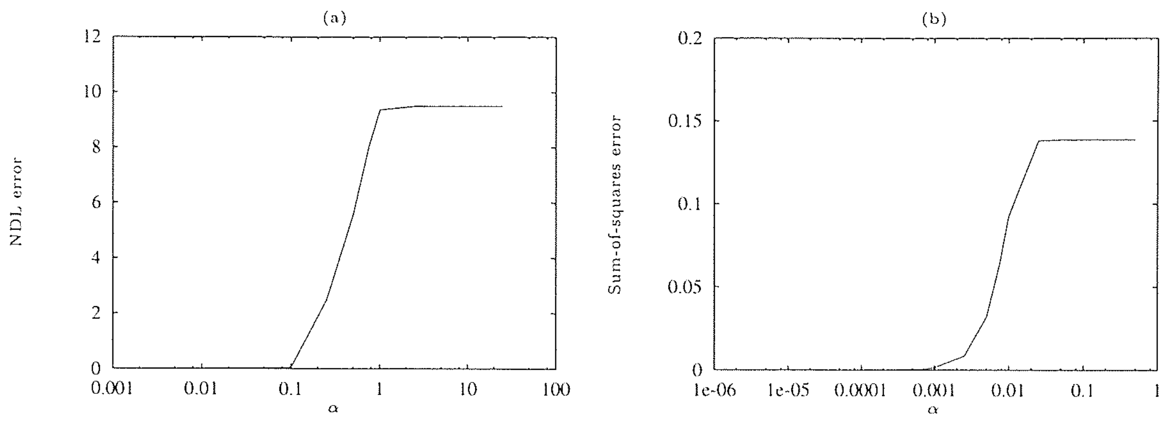

Figure 3.18: The two-lines training set α-surveys for differently trained networks. (a) The NDL data error of different values of the regularising parameter α, $\epsilon = 0.05$. (b) The training sum-of-squares data error.

Chapter 4.

The experiment just described showed the problem of a two-valued mapping in the data. The next experiment shows that the 'snapping' effect can also occur for 3-valued mappings. The training data shown in Figure 3.19a shows three horizontal clusters of points. The outputs are 0.1, 0.3 and 0.9 respectively. The network is required to discover 3 levels of structure in the data. This is made certain by the number of points in the bottom set. The number of points in each cluster is different where the bottom cluster contains 6 points, the middle cluster 3 points and the top cluster 2 points.

A network similar to the one used in the last experiment was used, except that 110 nodes were used in the hidden layer. Again it was important that the network be allowed to fit all the data for small values of α so the inputs were coarse-coded. The network should converge on the largest cluster of points for large α, in this case the bottom cluster of 6 points. This continued until the value of α was reduced to 0.75 as shown in Figure 3.19a.

When the value of α was reduced to less than 0.75 the network fit the bottom two clusters of points at output 0.1 and 0.3. As α was decreased even more the network continued to fit only the bottom two clusters of points as shown in Figure 3.19a.

When the value of α was 0.001 the network fit all the data as shown in Figure 3.19b. Even though a small value of ϵ will favour those outputs that lie within the ϵ scale the points in all the clusters can still be fit by the network solution because the weight space is large enough for all the points to be fit.

The error for each value of α was again plotted. The regularisation parameter α has been shown to control the 'stiffness' of the model and affects the measure of the number of well-fit points. A model that is slightly too stiff, fitting a few points, will have a

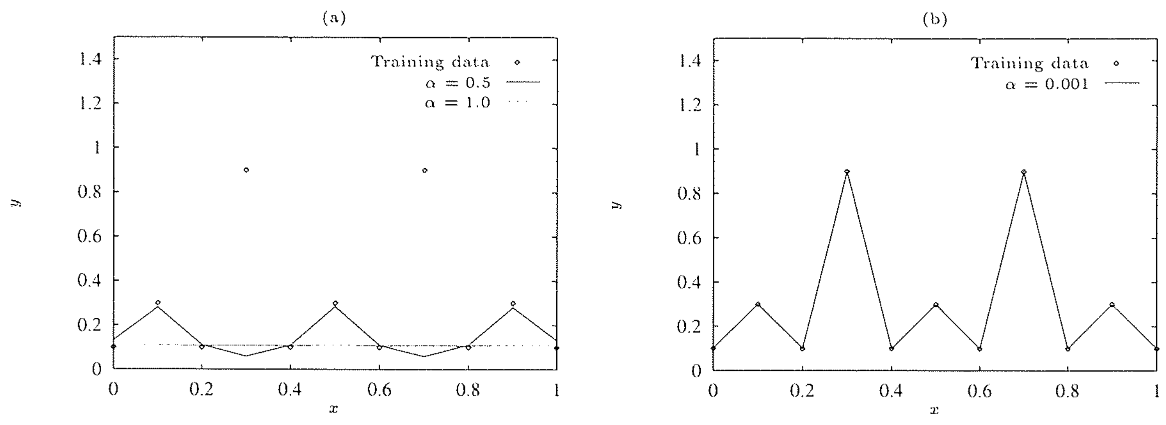

Figure 3.19: The three-lines training set and network outputs for different values of α. (a) When the value of α is high enough the points at $y = 0.9$ are ignored and all other points are fitted. The value of α is high enough to allow only the lower set of inputs at $y = 0.1$ to be fitted. (b) When the value of α is 0.001 the network solution fits all the data.

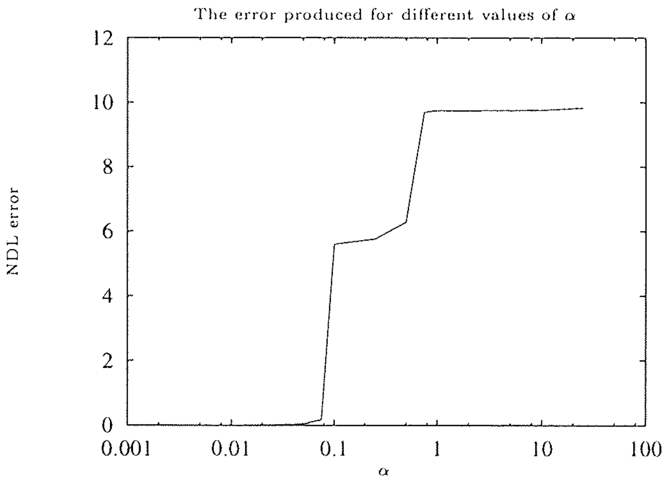

Figure 3.20: This graph shows how decreasing α affects the error on the three-lines training set. At certain values of α the error falls sharply as more training points are fit by the network.

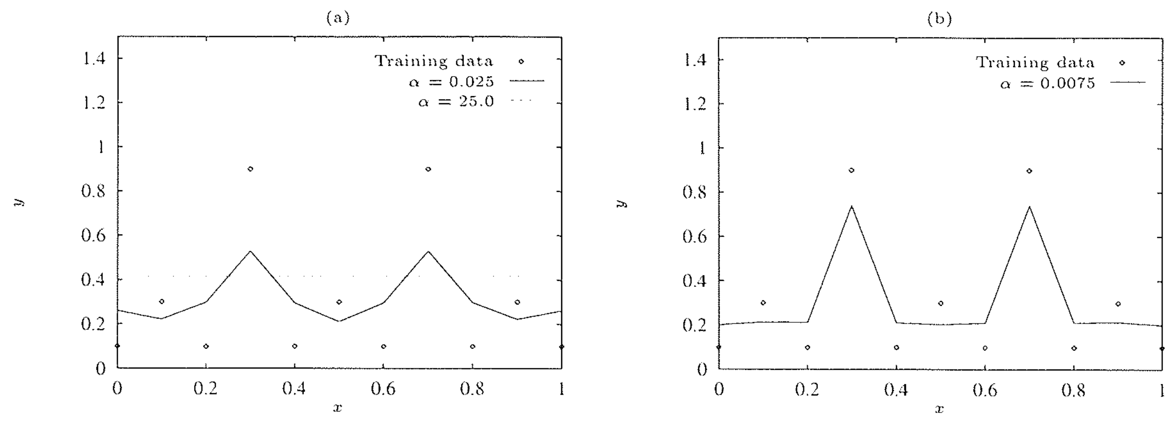

Figure 3.21: Outputs for networks trained using the sum-of-squares cost function on the three-lines training set. (a) Outputs when $\alpha = 0.025$ and $\alpha = 25.0$. (b) Outputs when $\alpha = 0.0075$.

larger training error than a 'looser' model fitting more points. The sudden decreases in error correspond to the places where the number of points that the network can fit changes rapidly. The sudden decreases or 'cliffs' that occur as a result of a change in the number of points fitted is shown in Figure 3.20. The graph shows the different levels of structure that exist in the data. A value of α would be chosen at the base of each 'cliff' that represents the most regulated fit for a particular level of structure.

Training with the conventional sum-of-squares cost function over a similar range of values for α gives quite different results. For high values of α the network fits none of the points shown in Figure 3.21a. Even as α is reduced the network again fits none of the points in Figure 3.21b. Only at very small values of α will the network fit all the points. Plotting the sum-of-squares error for each value of α shows less clearly the structure in the data.

3.5 Summary

The NDL cost function does approximate the mode of a given set of data as described by the simple problem presented in Section 3.3.4. This is because models that fit points within the ϵ scale are preferred to fitting points outside the ϵ scale. The decrease in cost of trying to fit a point within the scale of ϵ for a multi-valued input will always be more than the increase in cost of fitting another modal point outside the scale. The mode that the network finally converges to depends on the initial set of weights. The network will also converge to a cluster in a data set that contains the most points. For clusters of roughly equal points either mode can be chosen.

An unsupervised learning problem was presented to show what effect the value of

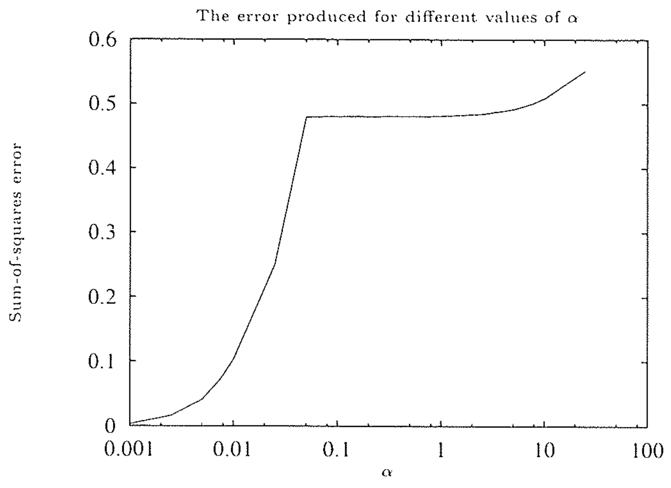

Figure 3.22: The sum-of-squares data error for a network trained with the sum-of-squares cost function on the three-lines training set. The regularisation α-survey does not show the same structure shown by using the NDL cost function in training.

ϵ would have on choosing a mode of data generated by a mixture of three Gaussian functions with different centres and variances σ. Starting at small values the densest cluster within the scale of ϵ was chosen first. As ϵ increased the number of 'well-fit' points increased.

A simple problem describing a data set containing a random part and a deterministic part was presented. The NDL cost function successfully identified and chose one of the values of the multi-modal mapping. By choosing different values of ϵ, the difference between the value chosen by the network and the actual value in the data was reduced.

The NDL cost function was applied to the inverse kinematics of a robot arm. The NDL cost function successfully solved the robot arm up or down problem. However some large errors occurred for positions where the robot arm had to suddenly flip orientation to fit the remaining end effector positions. The use of RBF networks greatly improved results. RBF's would allow a full α-survey to be done to achieve the best results. Neural Networks model continuous functions. This contributed to the large error on a small number of points for the robot arm problem. This problem was further illustrated by training a network to interpolate a noisy inverse $sin(x)$ function. The areas where the continuous function had to model discontinuous parts of the data could be seen.

The NDL cost function was found to work with contrived toy problems where there were multi-modal features in the data set. The two-lines and three-lines problem were toy problems where the use of regularisation was required to find the structure in the data. The structure in this case was represented by 2 and 3 clusters for the two-lines and three-lines problems. By generating an α-survey of these training sets the structure was found by identifying 'cliffs' or steps in the training data error. These results were compared with sum-of-squares training, which showed less clearly the structure in the data.

Chapter 4

Using NDL in regression problems

4.1 The α-survey

Many of the experiments performed in this thesis involved generating a regularisation α-survey of the training data. This involved setting the regularisation parameter α to a large value guaranteed to make a network underfit the data. This will typically drive the weights to very small values if weight-decay regularisation is used. When the network has finished training at a single value for α, the weights are stored and the value of α decreased. The network is then trained again with the new value of α and the weights initialised to the weights of the last network trained. Weight-decay using the square of the weights requires a large range for the regularising parameter α to capture the transition from underfitting the data to overfitting the data. For these experiments a network was required that would fit all the data. The data error had to be as close to zero as possible so that the network would fit all the data. This was so that all the different levels of structure could be seen. When plotting these different errors for the range of α it was required to use a logarithmic scale for plotting these α-surveys. The number of different values used for the regularisation parameter α was also important to capture the steps or 'cliffs' in the α-survey. The divisions of α were chosen to try to show the differences in error when clusters of data could no longer be fit by the model generated by using the NDL cost function. This was important to form a second derivative of the training α-survey. The method of starting with large α is preferable to starting in the opposite direction because of the saturation of sigmoidal units for small α complicating the increase in α.

The second derivative of the error with respect to $\log(\alpha)$ was calculated using the method of finite differences. The exact method is detailed in Appendix A.5. Because a large range of α was used the steps in α were calculated by taking the logarithm of the two extremes of α and calculating the range. This range was divided by the number of steps required and then exponentiated to get the division between two values of α. This was necessary to make the second derivative calculations simpler because the divisions in $\log(\alpha)$ would be the same. An alternative method of calculating the second derivative is to fit a polynomial to the network training data error and then take the second derivative of that. The use of Radial Basis Functions allowed networks to be trained very quickly. For all of these experiments the networks were trained until they reached a minimum. If the error did not decrease after a number of training epochs, training was stopped and the weights stored.

Care had to be taken in the α-survey that the weights were not all driven to the same value. This would happen for very large values of α. Therefore a starting point just above the top of the 'cliff' was chosen. This was done by training a single network with a large value of α and looking at the weights to see if large numbers of weights shared values that were approximately zero. If a network had weights that were all different then it was used as the starting set of weights at that value of α. The minimum training errors for each network were collected together to give the training set data error α-survey graphs. In a similar way graphs for the number of 'well-fit' points could be generated.

Testing the networks was done by loading the weights obtained for each value of α in training and using the test set to get a set of outputs. The sum-of-squares or NDL cost function was then used to assess the performance of the network and to generate the test set data error α-survey graphs.

4.2 The three-lines problem

In Section 3.4.5 it was shown that the levels of structure could be identified in a data set by using the NDL cost function. The regularisation parameter α could be chosen that corresponds to the most regulated fit at each level of structure. Calculating the second derivative of the training error for the three-lines problem shows that the second derivative peaks point to the different levels of structure in the data. The positions of the peaks correspond to the bases of the 'cliffs' in the training error. Figure 4.1 shows the network outputs for each network trained at different values of α. Beside the network outputs, the second derivative of the error and the error itself are plotted. The shaded areas show regions in the α-survey before and after the peaks in the second derivative. Figure 4.1a shows that the second derivative of the sum-of-squares training error does not indicate any changes in structure found. Figure 4.1b however has two peaks corresponding to the different levels of structure found using the NDL cost function in training.

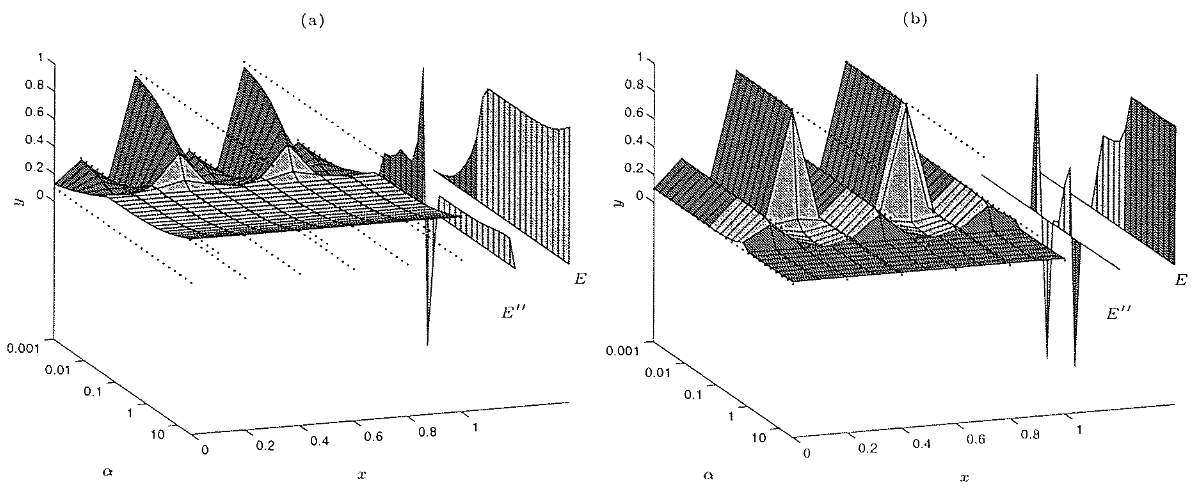

Figure 4.1: The results of training with the sum-of-squares and NDL cost function on the three-lines training set. (a) The network outputs, second derivative and sum-of-squares data error for a network trained with the sum-of-squares cost function. (b) The network outputs, second derivative and NDL data error for a network trained with the NDL cost function.

4.3 The noisy two-lines problem

Figure 4.2 shows a training set similar to the two-lines problem with many more points. Again it shows two clusters of data, generated by choosing random values for the x dimension using a uniform probability distribution between 0 and 1. The y values were generated by randomly selecting one of the two values 0.333 or 0.667. Gaussian noise was added with the mean being equal to one of the above values with a variance of 0.02. The aim of the following experiments is to show how the NDL cost function performs for a larger toy problem and to show other features of training and testing.

4.3.1 Sum-of-squares cost function training

Figure 4.3a shows the training sum-of-squares data error for the training set shown in Figure 4.2. As the regularisation parameter α decreases the network is able to fit more of the data points until it fits all the data when α is very small. Figure 4.3b shows the test set sum-of-squares data error. For large α the network mapping is a straight line between the two clusters of data. This is the best solution using the sum-of-squares cost function and so the sum-of-squares data error at this point is smallest.

The same network trained with the sum-of-squares cost function was tested with the NDL cost function for different values of ϵ. Figures 4.3c, 4.3d, 4.3e and 4.3f show the NDL data errors on the test set for different ϵ. Different values of ϵ gave quite different results for the error at different scales. When ϵ is quite large at $\epsilon = 0.1$

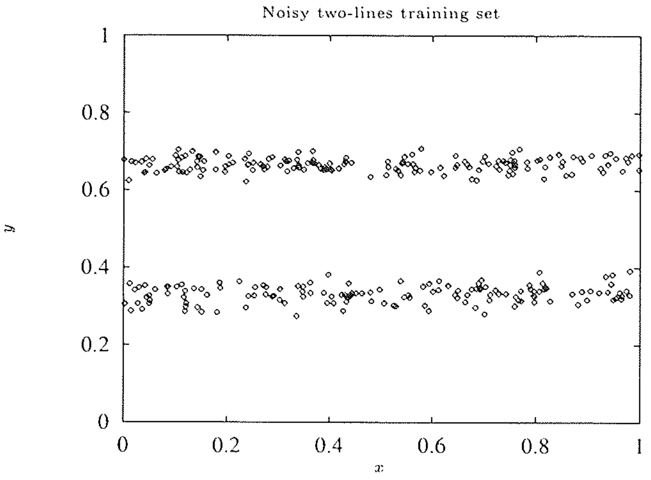

Figure 4.2: A training set containing random points generated from two Gaussian functions, one centred at 0.333 and the other centred at 0.667. Both Gaussians had a variance of 0.02.

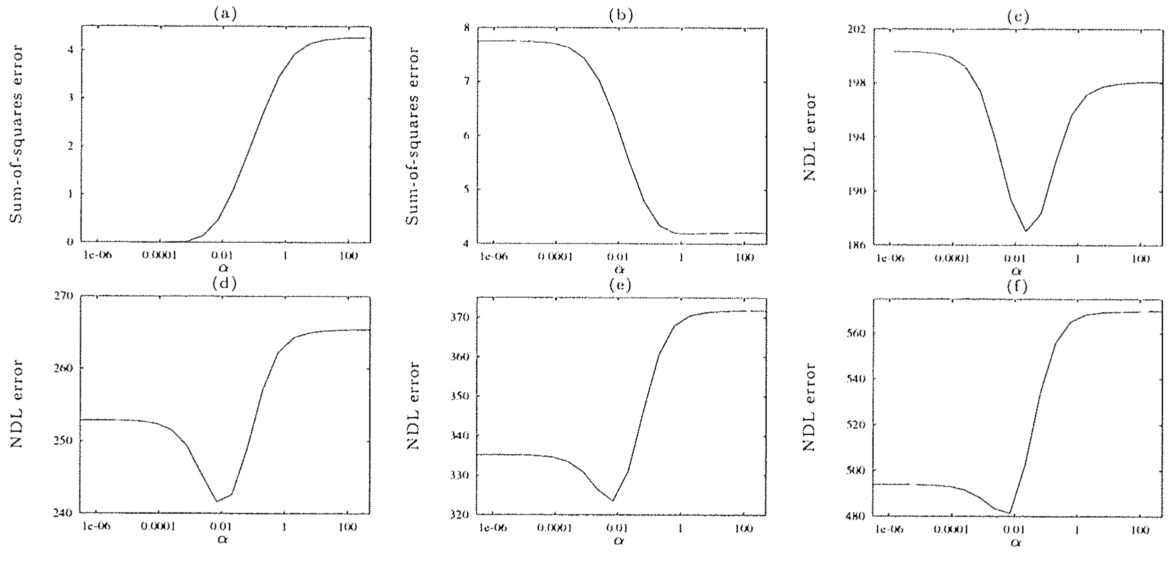

Figure 4.3: The results of training with the sum-of-squares cost function on the noisy two-lines training set. (a) The training set sum-of-squares data error α-survey. (b) The test set sum-of-squares data error. (c) The test set NDL data error with $\epsilon = 0.1$ for a sum-of-squares trained network. (d) $\epsilon = 0.075$, (e) $\epsilon = 0.05$ and (f) $\epsilon = 0.025$.

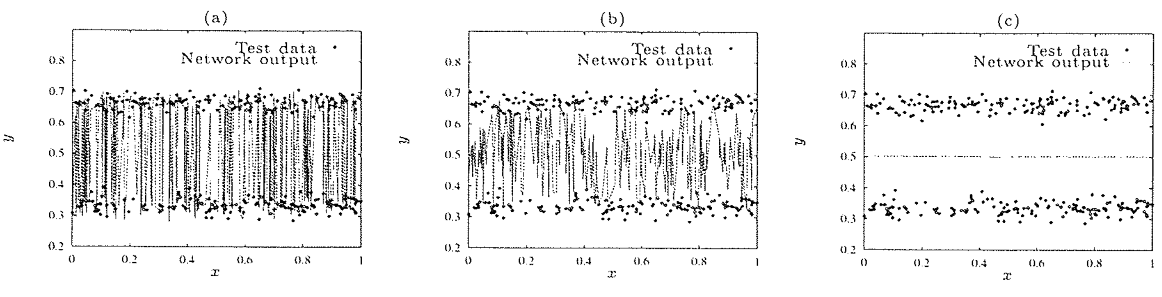

Figure 4.4: The outputs for networks trained using the sum-of-squares cost function. (a) When $\alpha = 0.00012$. (b) When $\alpha = 0.026$, (c) When $\alpha = 137.97$.

as in Figure 4.3c, the first part of the graph where there is little regularisation looks relatively similar to sum-of-squares testing in Figure 4.3b. Errors made on the test set are significant within the ϵ scale. The network overfits the training data and the errors are due to the different data used in the test set. Figure 4.4a shows the network outputs when α is small. The minimum test error occurs when $\alpha = 0.026$ in Figure 4.4b and the network has not overfit the points. As the network becomes highly regularised the network can no longer fit points within the ϵ scale as the network outputs become the mean of the two clusters. The NDL data error rises sharply when this occurs, see Figure 4.4c. In Section 3.3.5 the number of 'well-fit' points was described where the points fit by a network could be calculated for a small value of ϵ. When sum-of-squares cost function training is performed with a large value of α the error increases. This corresponds to a large fall in the number of well-fit points. The use of smaller values of ϵ in testing showed this effect more clearly as the relative increase in error for highly regularised networks was larger as ϵ became smaller.

Notice also that the minimum test set NDL data error occurs approximately at the same value of α. When training with other regression problems similar to this, the same NDL α-surveys can be generated. For most of the problems presented so far the separation between the clusters in the training data was constant and the data lied along some clearly defined region. This is shown in the example above, which is an extension of the simple two and tree lines problem. Here there are two noisy generators of data. Training with the NDL cost function should be able to detect the structure in this simple problem. Later in this chapter other problems will be described where there are no clearly defined clusters.

4.3.2 NDL cost function training

Networks that were trained with the NDL cost function and tested using the NDL and sum-of-squares cost function are shown in Figure 4.5. The networks were trained

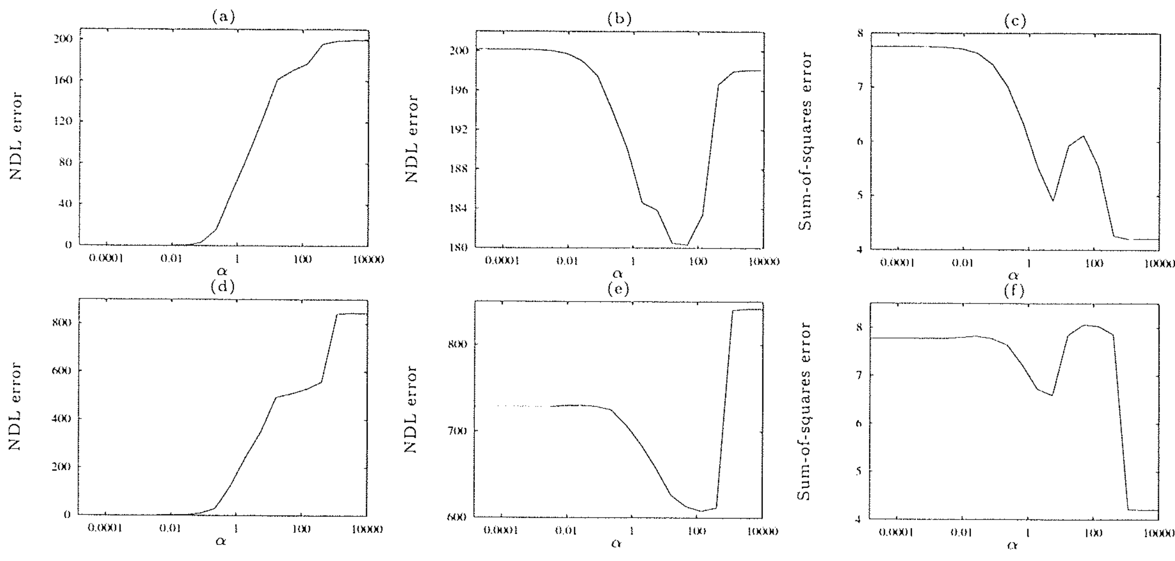

Figure 4.5: The noisy two-lines train and test set data errors for the noisy two-lines data set. (a) The training NDL data error with $\epsilon = 0.1$ (b) The test set NDL data error with $\epsilon = 0.1$. (c) The sum-of-squares data error for a network trained using the NDL cost function, $\epsilon = 0.1$. (d) The NDL cost function training α-survey with $\epsilon = 0.01$. (e) The test set NDL data error with $\epsilon = 0.01$. (f) The sum-of-squares data error for a network trained using the NDL cost function, $\epsilon = 0.01$.

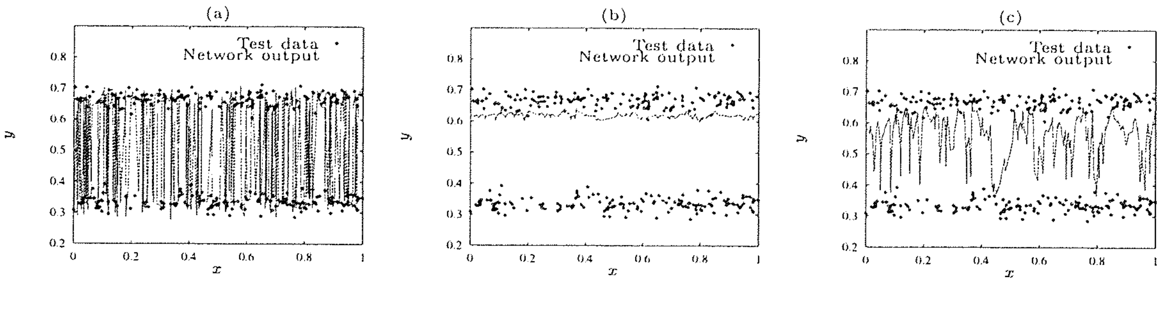

Figure 4.6: The outputs for networks trained using the NDL cost function. (a) When $\alpha = 5.55$. (b) When $\alpha = 47.28$, (c) When $\alpha = 0.0001$.

with two values for ϵ, 0.1 and 0.01. For a network trained with $\epsilon = 0.1$ the training α-survey in Figure 4.5a looks similar to the training α-survey using the sum-of-squares cost function in Figure 4.3a apart from a small kink around an α value of 100. This is where the network model can no longer fit the mean of the two clusters of data and moves to one portion of the data inside the scale of ϵ as α is decreased. This is more prominent in Figure 4.5d where two definite steps can be seen. These steps occur between a network that does not fit any points by converging to the mean, a network that fits just one cluster of points and finally a network that fits all the points.

Figure 4.5b and 4.5e show the NDL data error for the test set. Notice that the test set data error is not at a minimum when training has overfit the whole of the data in Figure 4.6a. Neither is it a minimum just before the training error decreases rapidly as α decreases where few of the points are fitted in Figure 4.6c. The minimum test set NDL data error occurs just after the training NDL data error decreases rapidly as α decreases to fitting one of the cluster of points in Figure 4.6b. It is noticeable that the base of the 'cliff' in the NDL data error during training, where $\alpha = 402.6$ in Figure 4.5d, does correspond approximately with the minimum test set NDL data error. This suggests a good indication of where to set a regularising parameter by looking at the training NDL data error alone.

The sum-of-squares data error was calculated for the test set for each network and is shown in Figures 4.5c and 4.5f. For large values of α the network has converged to the mean of the two clusters of noisy data and here the error is small. When the regularising parameter α reaches a value of about 1000 the sum-of-squares data error increases rapidly as α decreases. Figure 4.5f shows a peak when α reaches 100. This large error corresponds to the NDL cost function trained network converging on one single cluster of data in Figure 4.6b. As α decreases more of the points within a particular cluster are fit by the network model and so the sum-of-squares error decreases. When the value of α is small enough for all the points to be fit then the error corresponds to the difference between the randomly generated points in the training and test sets. As the value of ϵ is increased it will begin to look more like the test set data error for sum-of-squares training, see Figures 4.5c and 4.3b.

The noisy two-lines problem showed the effects of NDL cost function testing on networks trained using the sum-of-squares cost function. It also showed the effects of sum-of-squares cost function testing on networks trained using the NDL cost function. These tests alone may give some insight into what type of data is being dealt with and the structure within it. Many problems give sum-of-squares test errors that have large errors for underfitting and overfitting and relatively small errors between these two extremes. Figure 4.3b shows quite different results for multi-modal data. When training is performed with the NDL cost function the test set sum-of-squares data error

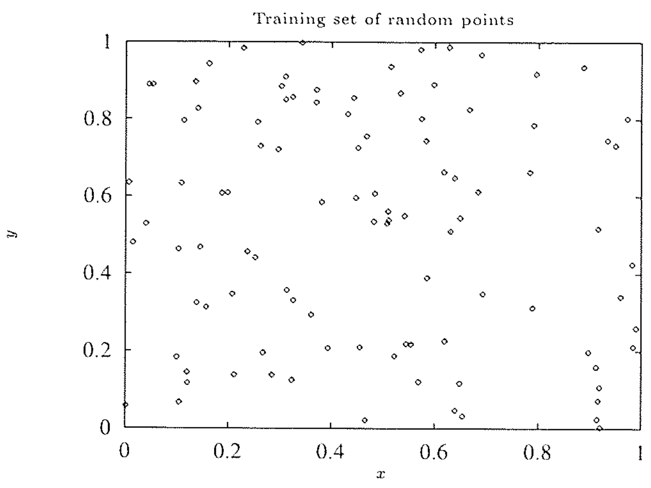

Figure 4.7: 100 random points generated from a uniform probability distribution between 0 and 1 for each dimension.

is relatively large. This is because the network has converged on a single cluster of data as shown in Figure 4.5c and 4.5f.

4.4 Practical considerations of the α-survey

The 'cliffs' that have been shown in previous α-surveys arise because of structure at different scales. To show that the α-survey can be smooth over a range of α values a random data set was generated shown in Figure 4.7. Each value for x and y was chosen from a uniform probability distribution between 0 and 1. To generate an informative α-survey for a particular problem the range of values must be chosen correctly as already mentioned in Section 4.1. The number of divisions within this range is also important. Too small a number of divisions may miss out 'cliffs' in the training α-survey. For using the second derivative of the training error to pick a value of α for regularisation a fairly coarse set of values are required to get an approximation to a good value for α. Using too many values will increase training times. Small changes in α will tend to give irregular changes in the number of points fit by the model. The α-survey may look smooth when viewed as a whole but may have many small 'kinks' when looked at more closely.

Figure 4.8a shows how the sum-of-squares data error changes in a training α-survey. Figure 4.8b shows the NDL data error during training. This is not as smooth, with slight differences due to the possibility that a few of the points will form a small cluster in the data. As the number of points increases in the data set, it will look more like 4.8a. Figure 4.8c shows the same network trained with twice the number of values of α

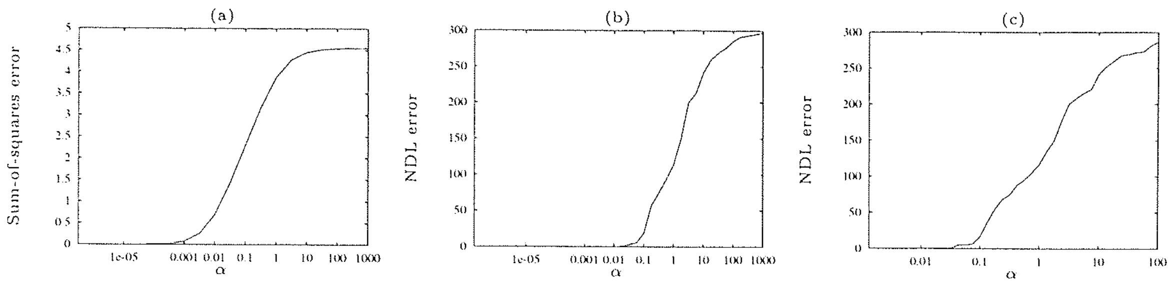

Figure 4.8: The training α-surveys for differently trained networks on the random training set. (a) A network trained with the sum-of-squares cost function. (b) NDL cost function training with $\epsilon = 0.01$. (c) NDL cost function training with twice the number of values of α in the α-survey. The range of α was also reduced.

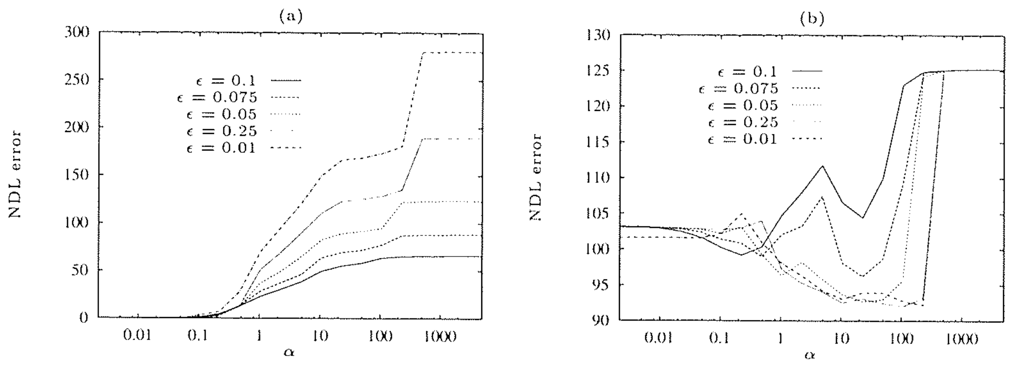

Figure 4.9: The effect of varying ϵ in NDL cost function training on a noisy two-lines training set. (a) The training NDL data error. (b) The test set NDL data error.

used in the α-survey. A smaller range of α is chosen in this case. It may be thought that using smaller steps in α may pick up local 'cliffs' but the α-survey is generally smooth. Using a higher resolution of α does not affect the α-survey in this case. However, the small local differences will seriously affect the interpretation of second derivative plots.

A better method of generating α-surveys than the ones presented here may be to decrease the size of the steps in α if the training error decreases rapidly. This would also improve the quality of any second derivative calculations.

4.5 Regularisation and the effect of ϵ

It has already be shown that through regularisation, steps or 'cliffs' can be seen in the NDL data error training α-surveys. It has not been shown whether different values of ϵ cause these steps to occur in the same places. The last section on the noisy two-lines problem was not conclusive in the case of NDL training. Data sets were generated in

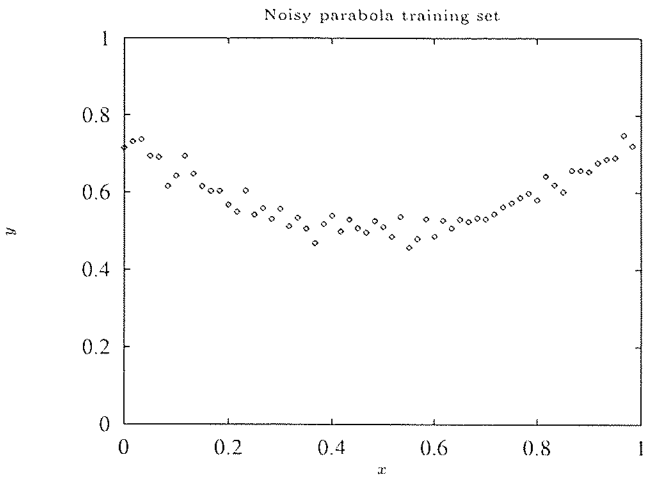

Figure 4.10: The noisy parabola training set of the form $y = (x - 0.5)^2$ with added zero mean Gaussian noise with variance of 0.02.

a similar fashion for the noisy two-lines problem described in Section 4.3. A data set was generated with 100 points. The aim of this experiment was to see how changing ϵ would affect the placement of the steps or 'cliffs'. Figure 4.9 shows the results from generating α-surveys with different values of ϵ. Figures 4.9a and 4.9b show the NDL data error training and testing results for different values of ϵ on 100 points. When ϵ is large at 0.1 the steps are not clearly seen. However there are slight changes that occur for α values of 100 and 10. As ϵ gets smaller the location of the bottom of the 'cliffs' does not change appreciably. The NDL data test set error for each of the networks trained with different values of ϵ were examined. Figure 4.9b shows that the smaller the value of ϵ, the sharper the 'cliff' is in the test set NDL data error.

4.5.1 The noisy parabola problem

In the noisy two-lines problem the data was divided into two distinct regions. It was a simple matter to choose a value of ϵ in the NDL cost function to allow the two regions to be distinguished by the NDL cost function. The next example shows how the NDL cost function performs with a problem that does not consist of multiple regions of data. The problem presented here is a generator of data following a parabola of the form $y = (x - 0.5)^2$ between the region of $x = 0$ and $x = 1$. Zero mean Gaussian noise with $\sigma = 0.02$ was added to the target y values. Again for these experiments an α-survey was done by decreasing α in small equal steps. In this case the mean and the mode of the data are the same. The intervals of α used for the sum-of-squares and NDL cost function was different because the sum-of-squares cost function generally overfit the data at smaller values of α. An RBF network was used with the number of basis

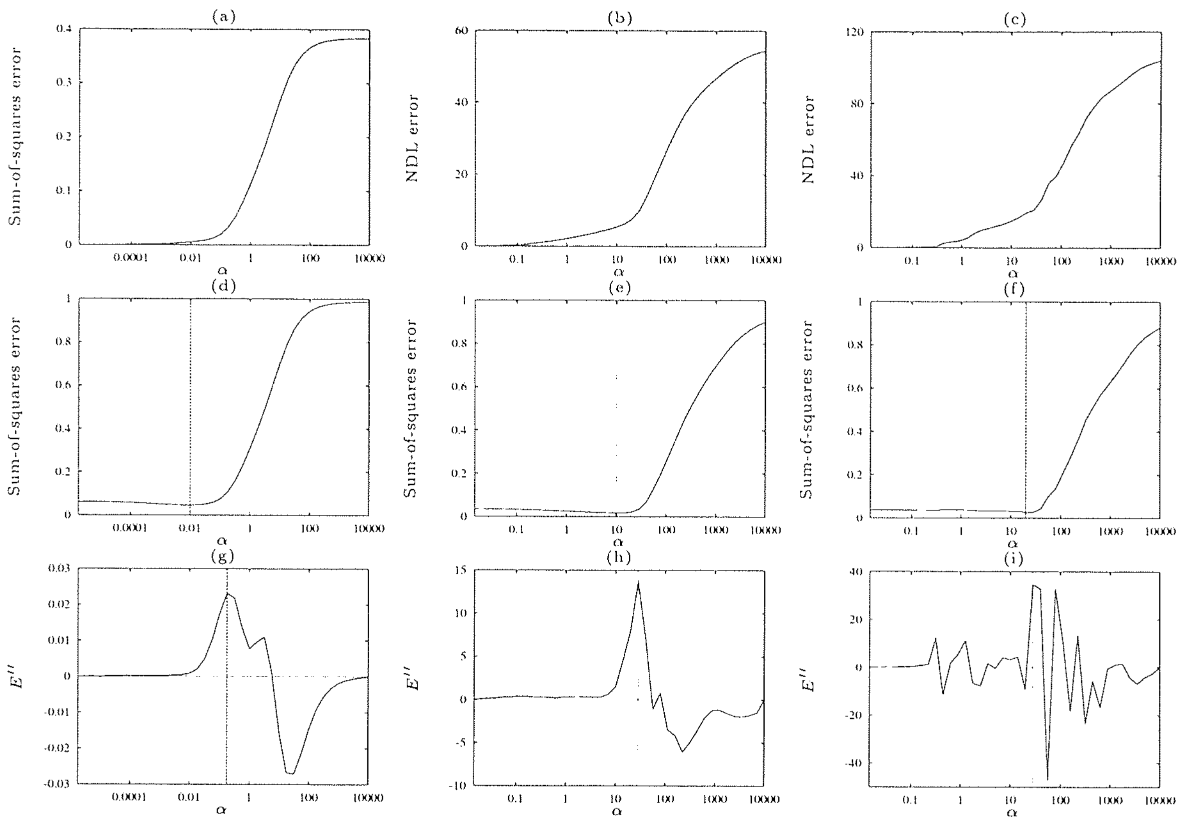

Figure 4.11: The training, testing and second derivatives for training with the sum-of-squares and NDL cost function on the noisy parabola training set. The vertical lines indicate the minimum test set error achieved and the maximum second derivative. (a) The training α-survey using the sum-of-squares cost function. (d) The test set data error for sum-of-squares. (g) The second derivative of the training sum-of-squares data error. (b), (e), (h) Similar results for NDL training with $\epsilon = 0.03$ and testing using the sum-of-squares cost function. (c), (f), (i) Similar results for NDL training with $\epsilon = 0.01$.

functions equal to the number of data points in the training set. Regression problems have already been presented where it was easy to choose a value of ϵ that would give good results. The results also showed clear steps or 'cliffs' in the training α-survey on which to set the regularisation parameter α. In this example it is again not so clear what the value of ϵ should be so different values of ϵ were used. Figure 4.10 shows a training set with 60 patterns. In this example the second derivative of the training NDL data error for each value of α was calculated. This could indicate a good value to set a regularisation parameter if only the training set were available.

Figure 4.11 shows a selection of the results obtained from training a network with the sum-of-squares and the NDL cost function for a number of different values for ϵ. The NDL data error training α-survey is shown for each and the test data error using the sum-of-squares cost function for each. The noisy two-lines problem clearly had

	Sum-of-squares			NDL, $\epsilon = 0.01$			NDL, $\epsilon = 0.03$			NDL, $\epsilon = 0.05$		
S	E''	E	D	E''	E	D	E''	E	D	E''	E	D
60	-1.73	-4.61	2.88	2.99	1.61	1.38	3.34	2.30	1.04	3.34	2.99	0.35
90	-1.73	-4.61	2.88	2.99	1.27	1.73	3.34	1.96	1.38	3.68	2.99	0.69
120	-1.73	-4.61	2.88	3.34	3.68	0.35	3.34	4.03	0.69	3.68	5.07	1.38

Table 4.1: The results of training and testing with the sum-of-squares and NDL cost function for different sized noisy parabola data sets. For each method of training the number of samples in the data set is shown. The value of α giving the maximum second derivative and the value of α giving the minimum test set sum-of-squares data error are given. The difference between these two last values is shown, $D = E'' - E$.

multi-modal data and the sum-of-squares test α-survey was different from the NDL test data error α-survey. For this problem there is no multi-modal structure in the generator of the data so the sum-of-squares cost function was chosen for testing. This raises the question as to when one knows when there is multi-modal data in a training set or not and whether the NDL cost function is best suited to highlight that underlying structure. This question is not answered here but would be a topic of further research. Besides the results shown in Figure 4.11 experiments were carried out with larger data sets using the same set of values for ϵ. These results are shown in Table 4.1. In Figure 4.11 vertical lines are drawn onto the graphs to show the minimum of the test set data error and maximum of the second derivative of the network error training α-survey. The aim is to show that the second derivative points to the best place to set a regularisation parameter. For training with the sum-of-squares cost function the second derivative does not point to the correct position in Figure 4.11g. When ϵ is quite large at 0.5 the second derivative points to a value close to the minimum test set NDL data error. When the value of ϵ is decreased even more the two match more closely. However at small values of ϵ the training NDL data error tends to highlight small local changes causing the training α-survey to look more jagged. This can be seen by looking at the second derivatives in Figure 4.11i. One solution already mentioned is to fit a smooth curve to the training data error α-survey and take the second derivative of that. The minimum test set data error for networks trained with the sum-of-squares cost function was greater than the minimum test set data error for both cases of NDL cost function training. Good results were obtained when the value of ϵ was less than the variance of the noise used in the generation of the data.

Table 4.1 shows similar results where α values corresponding to the minimum test set sum-of-squares data error and the maximum second derivative are recorded. The absolute difference between these two α values is also tabulated. One notable failure

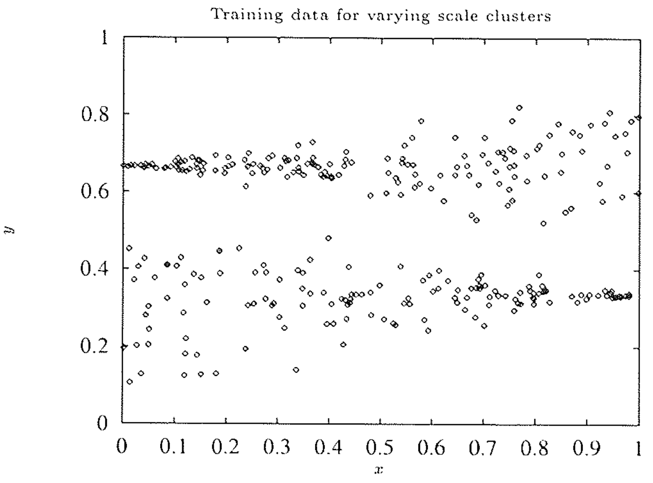

Figure 4.12: A data set showing a varying degree of width or scale in the two clusters of data.

was the case where the value of ϵ was 0.05 and the number of patterns was 120. Here the second derivative peak did not point to the minimum test set error. This may be due to local minima in the network training. These results support the use of the smallest possible value of ϵ.

4.6 NDL and variability in cluster scales

In Section 4.3 a noisy two-lines problem data set was described which consisted of multi-modal data. Figure 4.12 shows a data set where the scale of the two clusters of data vary. In this problem the network is still required to follow one cluster of data as in the noisy two-lines problem. The different scales within the data make the choice of ϵ more difficult. A large value of ϵ may cause the network to fit points from another cluster in the data. Another situation could also arise where the network output with a small value of ϵ may follow the top cluster of data where it is the densest. It may then drop down to follow the bottom cluster of data where it is densest.

The data was generated by choosing random values of x from a uniform probability distribution in the range 0 to 1. These points could either belong to the top cluster of data or the bottom with a probability of 0.5. If the point belonged to the top cluster then zero mean Gaussian noise was added to 0.66, with the width increasing linearly with x from 0. The opposite effect applied to the bottom cluster of data where noise was added in such a way that the width or variance of the zero mean Gaussian decreased with respect to the value of x.

Figure 4.13 shows a set of α-surveys with the data set shown. An RBF network

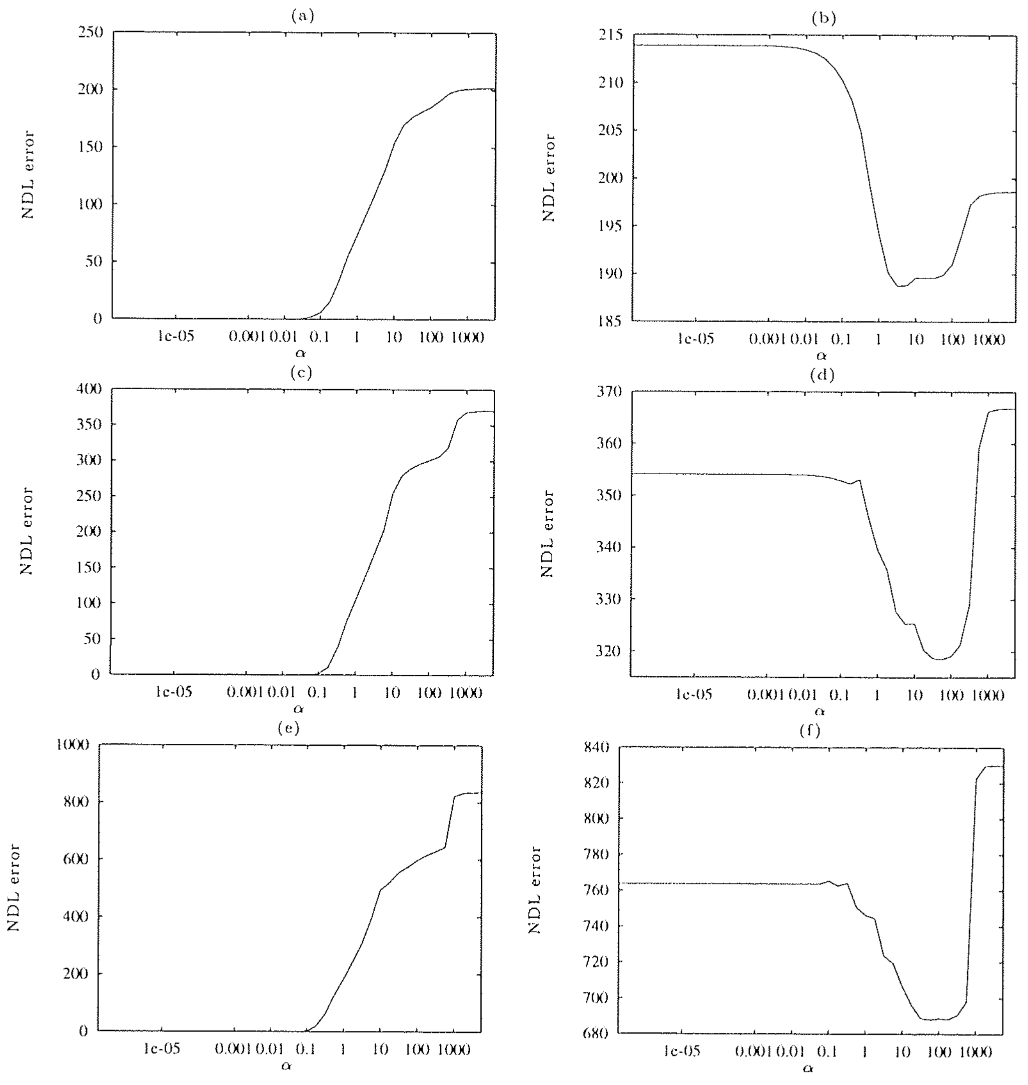

Figure 4.13: The NDL data error training α-surveys and test set α-surveys for the variable scale cluster data set. (a), (b) The results when $\epsilon = 0.1$ for NDL cost function training. (c), (d) The results when $\epsilon = 0.05$ and (e), (f) when $\epsilon = 0.01$.

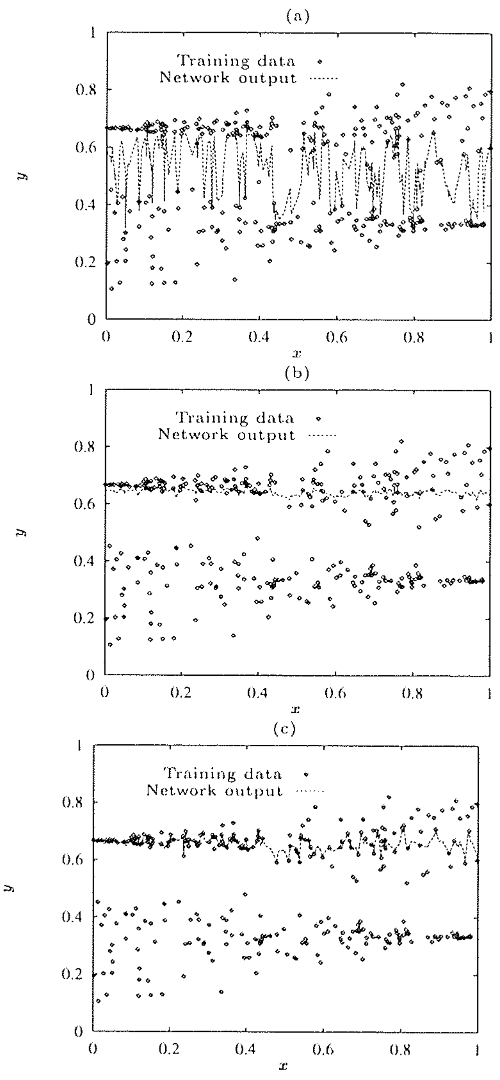

Figure 4.14: The network outputs compared to the variable scale cluster training set for networks trained using the NDL cost function. The network that gave the minimum NDL data error for the test set was chosen. (a) For a network trained with $\epsilon = 0.1$, (b) $\epsilon = 0.05$ and (c) $\epsilon = 0.01$.

with the number of basis functions equal to the 300 inputs was used. In each case a different value of ϵ was used, these were 0.1, 0.05 and 0.01. Again as ϵ is decreased the presence of 'cliffs' becomes more noticeable in the training α-survey.

The test set α-surveys show a minimum NDL data error at the base of these 'cliffs'. Figure 4.14 shows the network outputs for each of these values of ϵ. For each α-survey the minimum test set NDL data error was found and the network outputs were plotted compared with the original training set. When the value of ϵ was quite large at 0.1 the minimum NDL data error was found when the bottom points of the top cluster and the top points of the bottom cluster fit the model. This is shown in Figure 4.14a. As the value of ϵ decreased the network outputs followed more closely one of the clusters.

In Figure 4.14b the network outputs follow the top cluster of data. However where the points are the densest the network outputs have been drawn towards the bottom cluster of data. When ϵ is 0.05 some extra structure is seen in the data, signified by the presence of two steps in the NDL data error training α-survey. This occurs because the value of ϵ is small enough to follow only one of the clusters of data. When ϵ reaches a small value of 0.01 the network outputs are better for the densest parts of the data. However, where the data is less dense or more spread out the network outputs are more sensitive to the noise and thus tend to overfit some of the points in these regions, see Figure 4.14c. Even though the fit is not very smooth it may not be considered worse than Figure 4.14b for the less dense region. This is because it still stays within the cluster. In this sense, it still models the large scale structure. These results would tend to favour smaller values of ϵ in training.

4.7 NDL and variability in cluster 'separations'

In Section 4.3 a simple two-lines problem was described where a value for ϵ could be easily chosen to distinguish between these two output patterns. This section discusses problems where there is still a multi-modal feature, but the different modes occur at different 'separations'. Figure 4.15 shows two training sets that show clusters of data distributed at varying distance from each other. When x is 0 there are no multi-modal parts to the data. As x increases two clusters begin to appear. In this case the choice of ϵ is not so clear. The training set in Figure 4.15a was generated by picking a value of x at random between 0 and 1 and picking one of two functions describing parabolas, one bending down $y = -f(x)+0.5$ and one bending up $y = f(x)+0.5$ where $f(x) = (\frac{x}{1.6} - 0.05)^2$. Noise was added to the target values y consisting of zero mean Gaussian noise with a variance of 0.02. To highlight some of the problems that such a training set poses another training set was generated shown in Figure 4.15b. This set contains 150 pairs of values for x between 0 and 1 describing the same parabolas. For

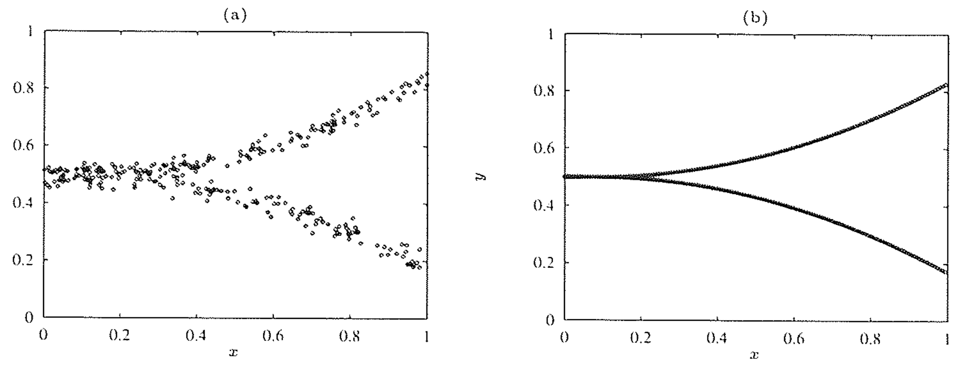

Figure 4.15: Data sets showing multi-modal data at different 'separations'. (a) Data with noise along two parabolas, one bending down and one bending up. Zero mean Gaussian noise is added to the targets. (b) Multi-modal data generated for the two parabolas for each value of x.

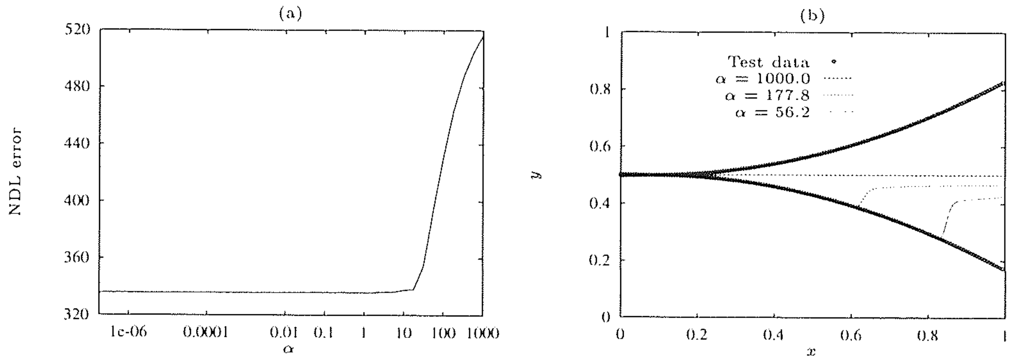

Figure 4.16: The results of training using the NDL cost function on the variable separation data set. (a) The training set NDL data error α-survey, $\epsilon = 0.01$. (b) Network outputs for different values of α in the survey compared to the training set.

each value of x the corresponding y values were calculated for both parabolas.

For the data set shown in Figure 4.15b an RBF network with 150 basis functions was used. An α-survey was generated and the training NDL data error is shown in Figure 4.16a for training with $\epsilon = 0.01$. In Section 4.3 a noisy two-lines problem was described where the two clusters of data are the same distance apart. The range of α where all the points could be fit by the model to one that only fit half the points was very small. However, for this experiment, we would expect a network to fit a modal point where the two values are closer to each other first, if α was decreased from a large value towards a small value. As α decreases in the survey more modal points are fit that are at greater separations. This is indeed what happens in Figure 4.16b. Firstly, the training α-survey in Figure 4.16a does not show such a sharp 'cliff'. Figure 4.16b

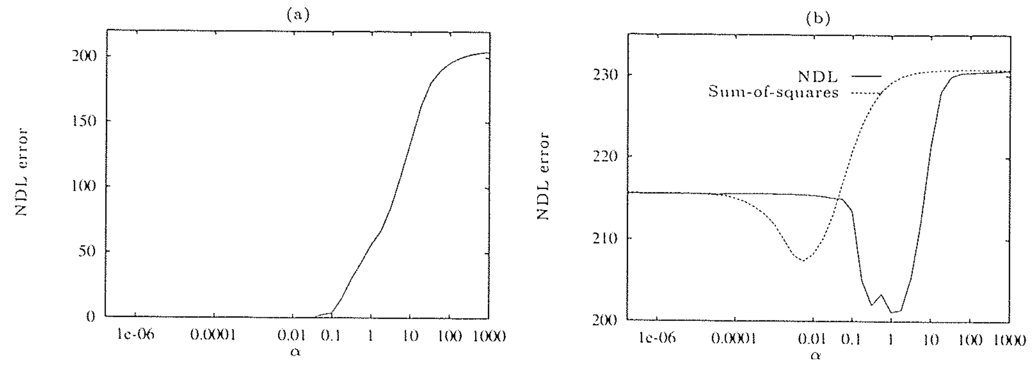

Figure 4.17: The results of training with the NDL cost function on the variable separation training set. (a) The training α-survey using the NDL cost function. (b) The test set NDL data error α-survey for networks trained with NDL and sum-of-squares cost function.

shows the network outputs for a number of different values of α. When α is 1000 the network output is approximately 0.5 for all values of x. As α decreases the network chooses to give an output corresponding to one of the two possible outputs. When α reaches a value of about 1 the network has chosen outputs corresponding to the parabola bending downwards in Figure 4.16b. As α decreases more of the points follow the bottom parabola.

The training set in Figure 4.15a shows the same type of data set, only the inputs are chosen at random between 0 and 1 and the outputs are generated that follow two parabolas with added noise. Again an RBF network was used, but this time 300 basis functions were used, one for each point in the data set. In this case a network was required to follow one of the two branches through regularisation. However the results for this problem indicate a deficiency in this method for data sets that have structure at different 'separations'.

Figure 4.17 shows the results after an α-survey was generated on the data shown in Figure 4.15a. The network was trained with both the sum-of-squares and NDL cost function with ϵ set to 0.05. The training NDL data error α-survey still showed rapid changes in the error but again this was not so pronounced, similar to Figure 4.16a. The training α-survey showed a smoother and more gradual change. Each network was tested with a separate set of data generated in the same way as the training set. Figure 4.17b shows the results of testing with the NDL cost function with $\epsilon = 0.05$. Figure 4.18 shows the network outputs for the NDL trained network where α was set at 1.8 corresponding to the minimum test NDL data error in Figure 4.17b. Closer inspection of this graph shows that in some regions the network outputs follow either the parabola bending down or the parabola bending up. However this approach could

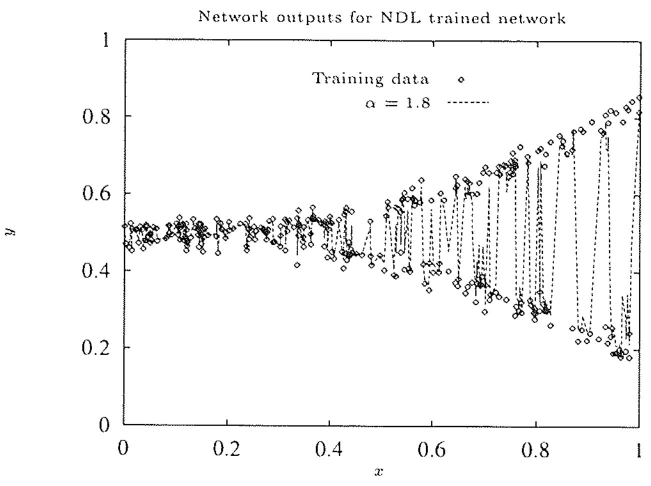

Figure 4.18: Network outputs and the variable separation training set for a network giving the minimum NDL test data error, trained using the NDL cost function.

be improved on greatly. Ideally the training α-survey would show two steps or 'cliffs'. One step where the network goes from fitting all the data to fitting one of the parabolas and another step when the network gives an average output of 0.5 for all the inputs. Despite the lack of the desired result the network trained with NDL did better that sum-of-squares training.

4.8 Using NDL on some real-life data

The NDL cost function was tested on some real-life regression problems to see if the training α-surveys could give some insight into choosing a value of α that would give good performance on a test set before it was seen. The NDL cost function was tested on the 1985 Auto Price, CPU performance and the Boston Housing data sets. These data sets were used by Quinlan [65].

4.8.1 Auto Price data set

The 1985 Auto price list data set contained 80 training patterns and 79 test patterns. Each pattern had 16 inputs and 1 target. An RBF network was trained with 80 basis functions. Figure 4.19 shows the results of training with the sum-of-squares and NDL cost functions. Training with the sum-of-squares cost function produced the smooth α-survey in Figure 4.19a. The second derivative of this survey showed a single positive peak. It did not however indicate a good value to set the regularising parameter α as the minimum test set mean-squared data error occurs at a smaller value of α. Training

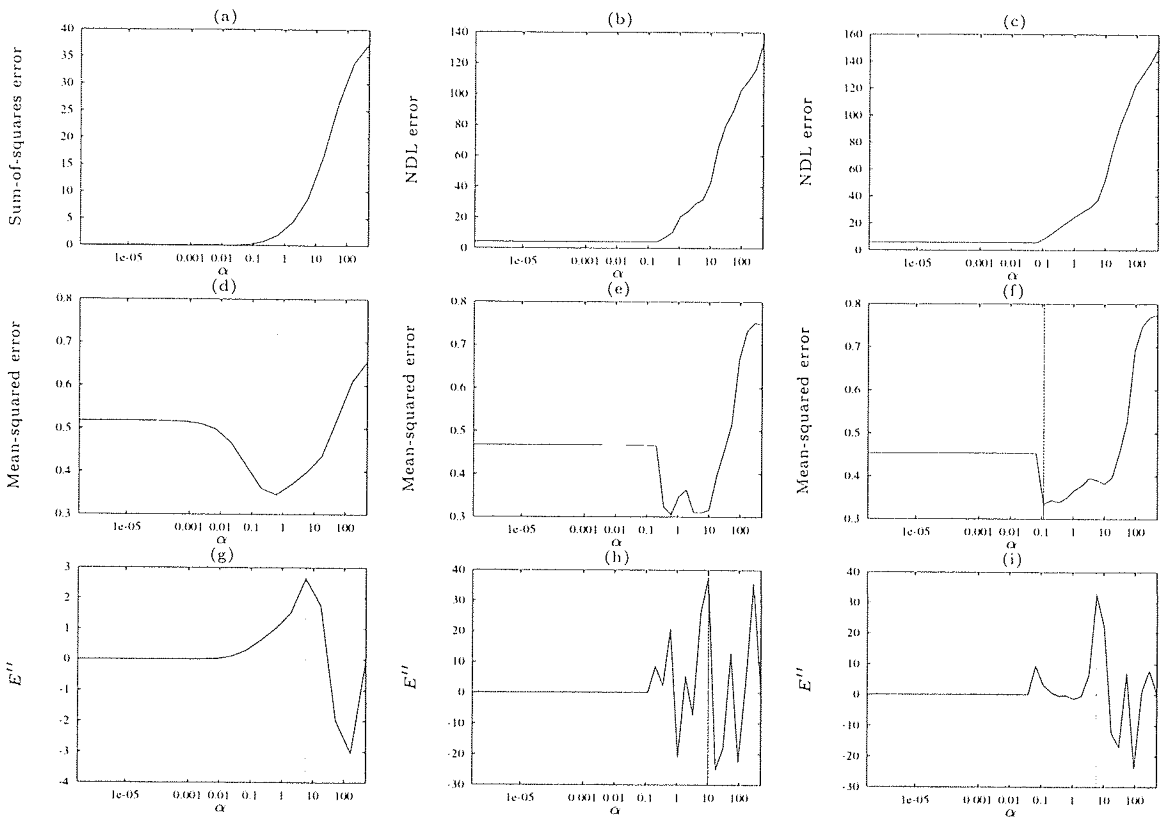

Figure 4.19: The results of training, testing and finding the second derivatives of the training α-survey for differently trained networks on the Auto Price data set. (a) The training α-survey using the sum-of-squares cost function. (d) The test set mean-squared data error. (g) The second derivative of the training sum-of-squares data error. (b), (e), (h) Similar results for NDL training with $\epsilon = 0.05$ and testing using the mean-squared cost function. (c), (f), (i) Similar results for NDL cost function training with $\epsilon = 0.03$ and testing using the mean-squared cost function.

with the NDL cost function has caused two steps in the α-survey, shown in Figure 4.19b when training with $\epsilon = 0.05$. These two steps are shown by the two peaks in the second derivative. These peaks point to two minima in the test mean-squared data error. The error at $\alpha = 0.6156$ of 0.3052 is slightly less than the error at $\alpha = 5.743$ of 0.3099. The largest peak does not in this case indicate the smallest mean-squared error. A second slightly smaller peak does point to the minimum test error. Training with the NDL cost function with $\epsilon = 0.03$ shows similar results. The two peaks in the second derivative of the training error point approximately to different minimum test errors in the test set α-survey. This presents a problem as to which peak should be used for setting the value of α where the data shows different levels of structure.

4.8.2 CPU data set

The CPU Performance data set contained 104 training patterns and 105 test patterns. An RBF network was trained with 104 basis functions. Figure 4.20 shows the results of training with the sum-of-squares and NDL cost functions. Training with the sum-of-squares cost function produced networks where the training data error approached zero for small values of α as shown in Figure 4.20a. However the test set mean-squared data error did not show a clear underfitting and overfitting performance as the Auto Price data set experiments. Figure 4.20d showed a minimum mean-squared error at $\alpha = 0.00891$ of 0.11582, indicated by the vertical line. The second derivative of the training α-survey showed a peak that did not point to this minimum test set data error. Training with the NDL cost function with $\epsilon = 0.03$ produced an α-survey with many small steps in Figure 4.20b. The minimum mean-squared error at $\alpha = 0.000141$ was 0.104, less than for sum-of-squares training. Figure 4.20f shows the second derivative of the training α-survey. There are many peaks pointing to the small steps in training, only the peak pointing to the first step in the α-survey is close to the minimum. Training with the NDL cost function with $\epsilon = 0.01$ showed similar results, however the minimum mean-squared error for the test set was much worse. This again shows the problem of which second derivative peak to use for setting the value of α.

4.8.3 Boston Housing data set

The Boston Housing data set contains data on the values of houses in the suburbs of Boston. There were 256 training patterns and 250 test patterns. An RBF network was trained with 256 basis functions using the sum-of-squares and NDL cost function. The number of inputs was 13 with 1 target. Only a small amount of overfitting occurred as the training data error approached zero for both methods of training in Figure 4.21a and 4.21b. Again the second derivative of the sum-of-squares cost function training

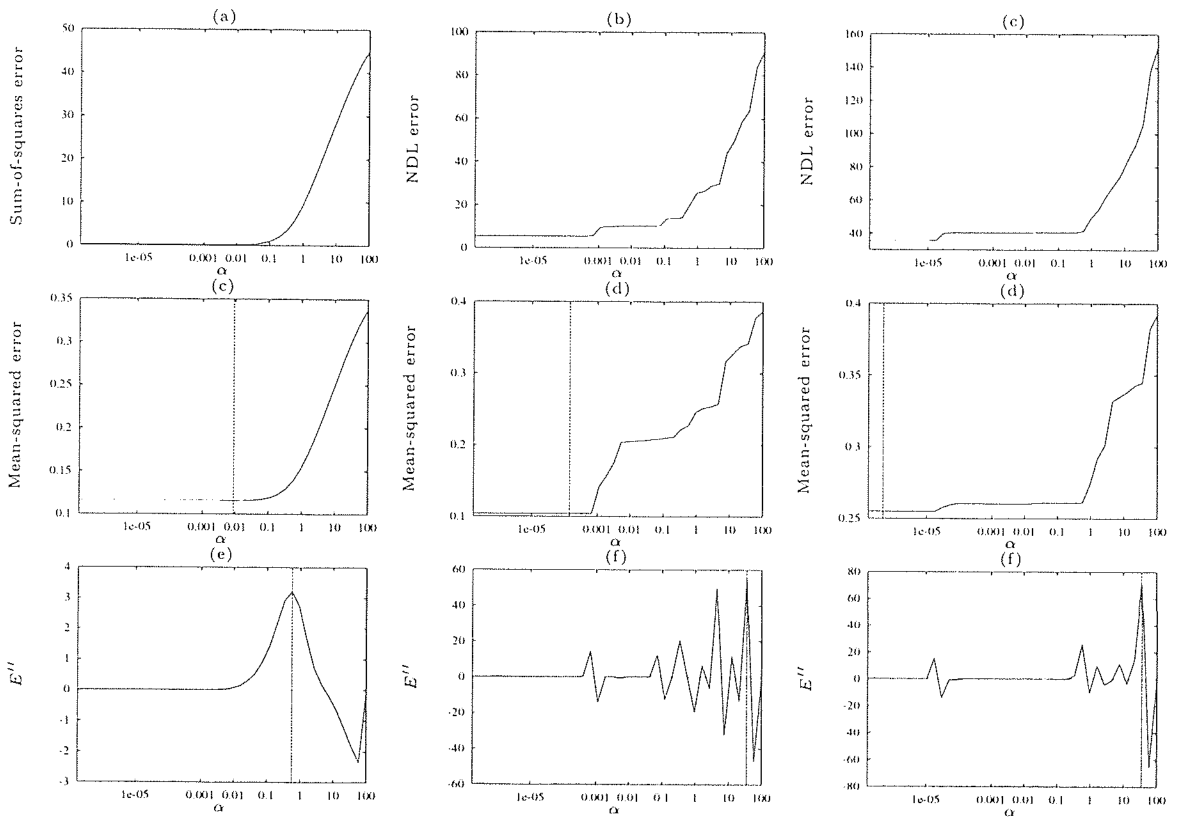

Figure 4.20: The results of training, testing and finding the second derivatives of the training α-survey for differently trained networks on the CPU performance data set. The vertical lines indicate the minimum test set error achieved. (a) The training α-survey using the sum-of-squares cost function. (d) The test set mean-squared data error. (g) The second derivative of the training sum-of-squares data error. (b), (e), (h) Similar results for NDL training with $\epsilon = 0.05$ and testing using the mean-squared cost function. (c), (f), (i) Similar results for NDL cost function training with $\epsilon = 0.01$ and testing using the mean-squared cost function.

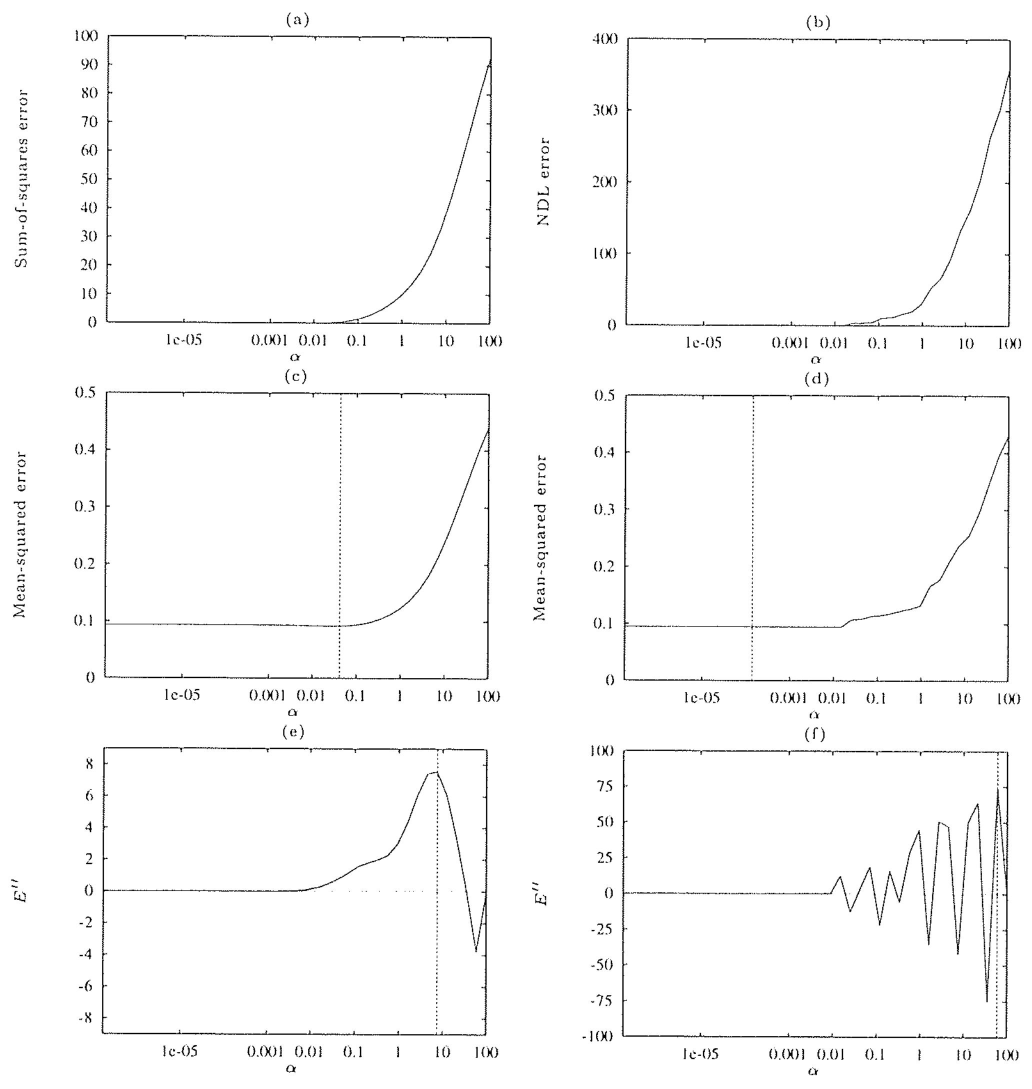

Figure 4.21: The results of training, testing and finding the second derivatives of the training α-survey for differently trained networks on the Housing data set. The vertical lines indicate the minimum test set error achieved. (a) The training α-survey using the sum-of-squares cost function. (c) The test set sum-of-squares data error. (e) The second derivative of the training sum-of-squares data error. (b), (d), (f) Similar results for NDL training with $\epsilon = 0.05$ and testing using the sum-of-squares cost function.

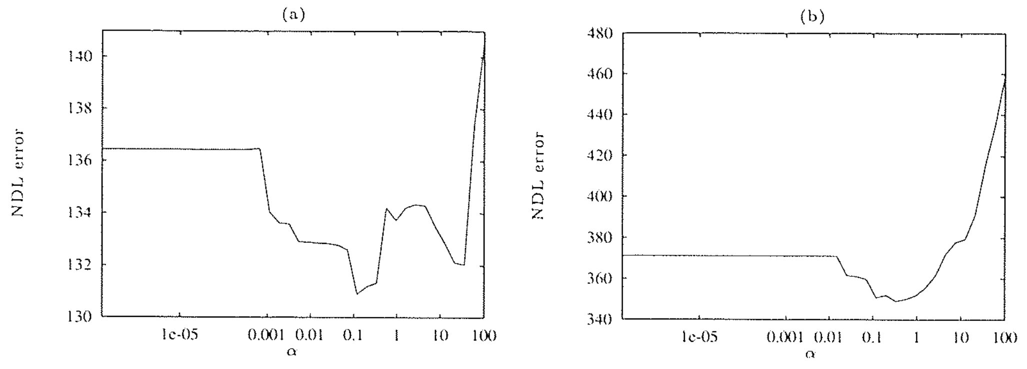

Figure 4.22: The test set NDL data error for different data sets. (a) The CPU performance test set, (b) The Boston Housing test set.

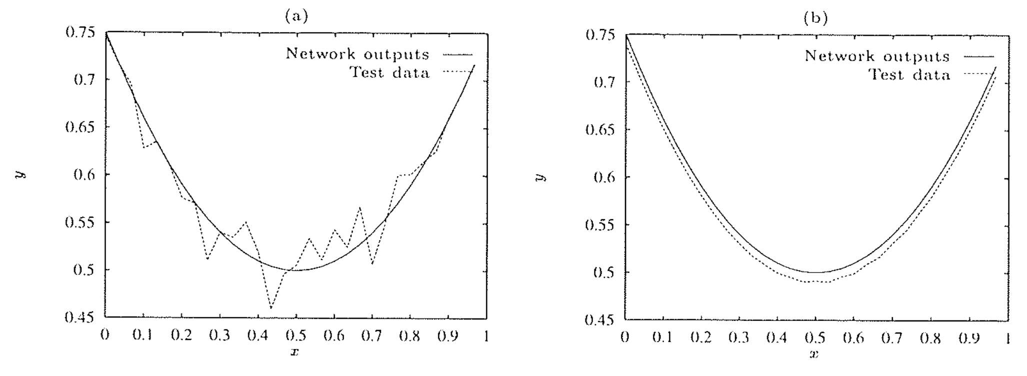

Figure 4.23: Different training sets compared to a test set describing a parabola. (a) A network trained with half the data describing a parabola with added random noise. (b) A network trained on data where a constant noise value is added to all data.

α-survey did not point to the minimum test set mean-squared error. The α-survey for training using the NDL cost function looked very similar in Figure 4.21b. The second derivative of the NDL cost function training α-survey did not show any peaks near the minimum mean-squared error for the test set.

A concern that may be expressed about the results for the CPU performance and the Boston Housing data set results was that the network did not properly fit all the data. Figure 4.22 shows that when the test error is measured with the NDL cost function there is a definite overfitting and underfitting performance.

4.9 Comparing NDL and Sum-of-squares

In the last few experiments it was clear that when measuring the test set mean-squared error for NDL trained networks the network did not appear to follow the overfitting and underfitting performance expected. When the test set error was measured according to the NDL cost function this feature was found. This difference is due to how the different cost functions interpret the errors. It has already been described in Section 4.3 that sum-of-squares cost function training did not produce a minimum test error between underfitting and overfitting for some data sets.

Figure 4.23 shows how the two cost functions of NDL error and sum-of-squares error can give two quite different interpretations of the data on which the network is trained. Figure 4.23a shows a network trained on data describing a parabola where half of the points are distorted by random noise. A network was trained using the sum-of-squares cost function to overfit the training data. The sum-of-squares error and NDL error were calculated for the test set that contained points along the parabola with no added noise. For the NDL cost function trained network the test set NDL data error would be small compared to the sum-of-squares error. This is because many of the points in the test set match the points in the training set and the NDL cost function greatly rewards points that can be fit within a certain scale. Figure 4.23b shows a training set where the points have all been distorted by a constant noise value. When this network was tested the NDL data error would be large as none of the points in the test set appear in the training set. The sum-of-squares cost function would give a small error compared to the NDL cost function.

For the CPU performance and Boston Housing data the number of points fit by the model during NDL training may have been constantly changing throughout the α-survey. Measuring the mean-squared error may produce no change because the particular points that the NDL cost function fits will produce no change in the mean-squared error.

4.10 Summary

The NDL cost function was applied to regression problems with the aim of being able to choose the regularising parameter α by examining what structure exists, if any, in the training set. For the three-lines problem the second derivative of the training data error α-survey showed two peaks pointing to the points where the 'cliffs' occurred. Second derivative calculations were made from the training α-surveys on the noisy parabola problem. The second derivative for sum-of-squares cost function training revealed a peak but this did not point to the minimum data error for the test set. The smaller the value of ϵ the better the results. This was the case until numerical difficulties in using the NDL cost function at very small values of ϵ in training caused these tests to be discarded. Potential problems of the use of the NDL cost function were shown. This included the training of networks on data sets that showed variability in the scale of the multi-modal clusters and in the separation of the multi-modal points.

Tests were performed on real-life regression problems with higher dimensional inputs. These showed good results where the peaks pointed to minima in the test set errors. They also showed multiple peaks pointing to local minima. This poses the problem of which one to use to obtain a minimum error over the test set before it is seen. Other problems of interpretation of NDL and sum-of-squares cost function test results were discussed.

Chapter 5

Using NDL in classification problems

5.1 Classification problems

Chapter 4 showed several features of the NDL cost function when it was used with regression problems. Generating training α-surveys with the NDL cost function showed 'cliffs' in the NDL data error indicating different levels of structure in the data. The second derivative of the NDL data error showed peaks corresponding to places in the training α-survey where the error changed rapidly. These peaks were shown to point to minimum data errors in the test set. This feature is now used in classification problems to choose a value of α from the training α-survey alone. For this purpose a simple classification problem is used to illustrate the principles. Later in this chapter real examples of classifying Vowel data and other classification problems are used. Results from NDL training have so far been compared to similar results using the sum-

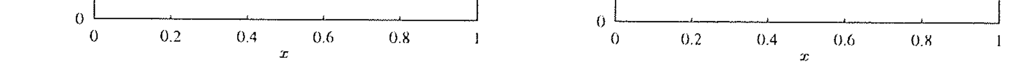

Figure 5.1: Simple classification problem with two classes. (a) Class one. (b) Class two.

of-squares cost function. In this chapter the NDL cost function is compared with the Cross-Entropy cost function. The Cross-Entropy cost function has gained a greater use in classification problems and is better principled because it corresponds to a multinomial noise model. For details on Cross-Entropy see Appendix A.4.

Figure 5.1 shows a classification problem containing two classes. Values were picked at random to form points in a 2-dimensional space. Each value was chosen according to a 0.5 mean Gaussian probability distribution in the x and y dimension with a variance σ of 1.0. Each point was classified on whether the x, y coordinate was inside or outside a circle centred on 0.5, 0.5, with radius 0.15. To illustrate the effects of noise, two methods of adding noise were used. In one method each point was miss-classified with a probability ϕ. The other method involved adding to each point some random noise according to a zero mean Gaussian with a variance of σ. Each training set contained 200 such points and a test set was generated in the same manner containing 2000 points. A number of different training sets and test sets were generated with different miss-classification probabilities ϕ and noise variances σ. Figure 5.1 shows a training set where $\phi = 0$ and $\sigma = 0$. The outputs of the network followed a simple 1-of-N coding scheme. If the point was within the circle it would be assigned to the output class $(1, 0)$ otherwise it would be assigned to the output class $(0, 1)$.

A network was trained using both the Cross-Entropy cost function and the NDL cost function for different values of the regularising parameter α, forming an α-survey. The value of α was decreased from a value guaranteed to give a rigid underfitting of the data to a value small enough for overfitting to be expected. When training was performed using the Cross-Entropy cost function on this simple classification problem the training data error decreased smoothly until the network fit all the data, see Figure 5.2a. Figure 5.2b shows the NDL cost function training α-survey. It shows two steps indicating different levels of structure.

Figure 5.2c and 5.2d show how the percentage of correct classifications compares for using Cross-Entropy training and NDL training with $\epsilon = 0.05$. Again, the aim of these experiments was to show that a value for the regularising parameter α could be found from the training set alone. For these experiments the performance was measured for the test sets to confirm whether such a choice based on the training α-survey would have given good test set results. Each of the test set points was input to the network and the outputs were calculated. As the classes were arranged as a 1-of-N coding, the class to which the input belonged was found by finding the output with the maximum value. For Cross-Entropy training the maximum percentage of correct classifications was 96.25% for a network trained with $\alpha = 0.022$. NDL training gave a maximum percentage of correct classifications of 96.2% for a network trained with $\alpha = 0.066$.

In Section 4.2 it was shown that the base of the 'cliff' in the α-survey should give

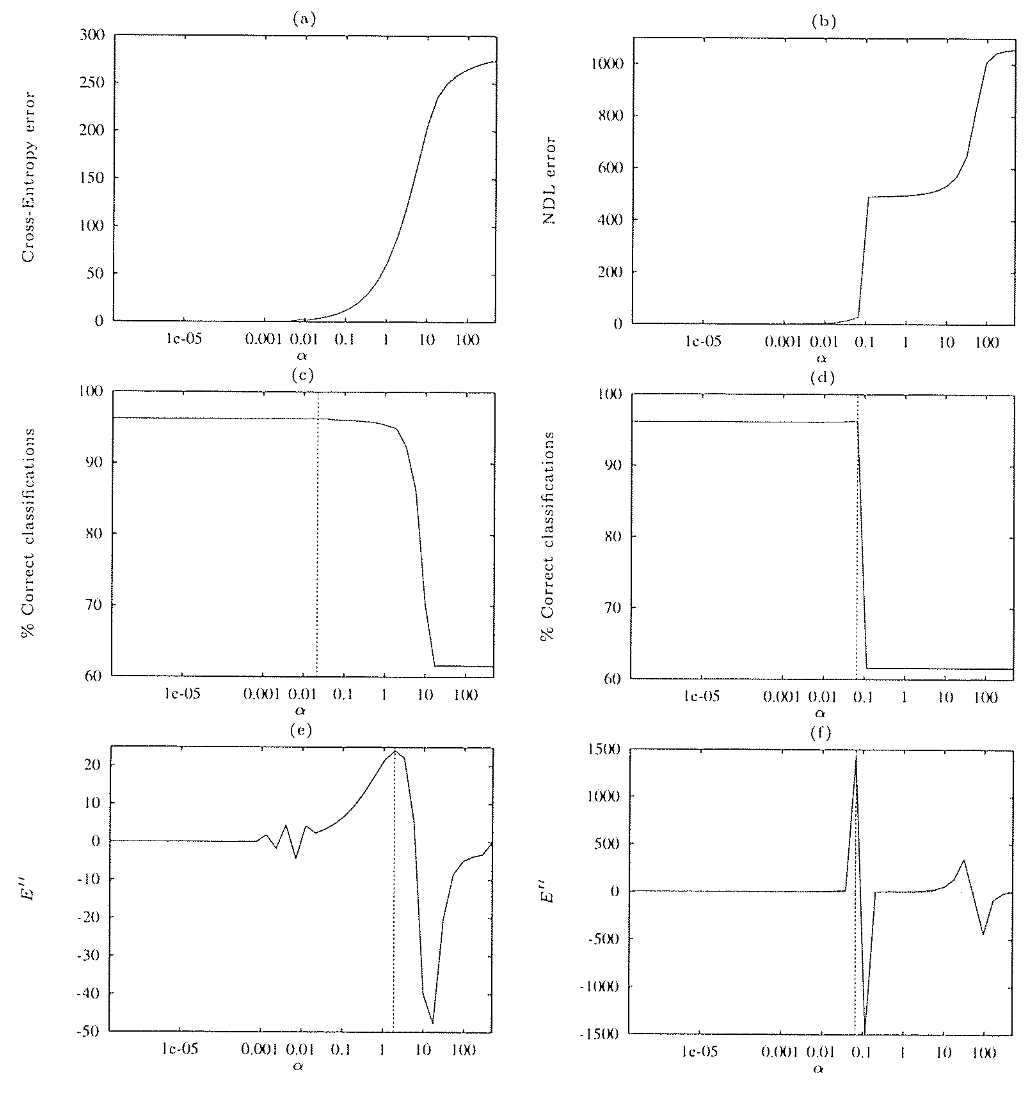

Figure 5.2: Cross-Entropy and NDL data error for training and testing on the two-class data set. The vertical lines indicate the maximum percentage of correct classifications achieved on the test set and the maximum second derivative. (a) The Cross-Entropy data error for training α-survey. (b) The NDL data error for training α-survey. (c) The percentage of correct classifications on the test set, for a network trained with the Cross-Entropy cost function. (d) The percentage of correct classifications on the test set, for a network trained with the NDL cost function. (e) The second derivative of the training Cross-Entropy data error. (f) The second derivative of the training NDL data error.

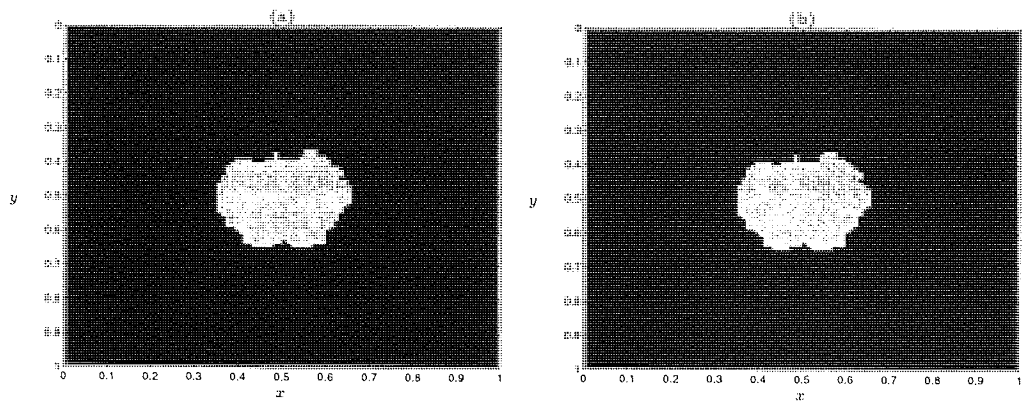

Figure 5.3: The two-class test set decision boundary. (a) A network trained with the Cross-Entropy cost function. (b) A network trained with the NDL cost function.

a good value to set the regularising parameter α before the test set is seen. A more precise method of finding the base of the 'cliff' is to calculate the second derivative of the error with respect to the value of $\log(\alpha)$. The second derivative gives the rate of change of the gradient of the training α-survey. In this case the second derivative will be large where there is a rapid change at the base of the 'cliff' and large negatively at the top of the 'cliff' where the gradient decreases rapidly.

Calculation of the second derivative of the Cross-Entropy training error with respect to $\log(\alpha)$ shows a peak at a value of $\alpha = 1.880$ in Figure 5.2e. This however does not point to the maximum test set percentage of correct classifications. The peak for the network trained with the Cross-Entropy cost function is broader and less defined than for the second derivative of the NDL training α-survey. Figure 5.2f shows a sharp peak in the second derivative at $\alpha = 0.0066$ pointing directly to the maximum test set percentage of correct classifications.

Figure 5.3 shows the decision boundary between the two classes for networks trained with the Cross-Entropy and NDL cost functions on a test set. The aim was to show that the decision boundaries will be similar for both methods of training. This test set contained a 100 by 100 matrix of numbers. Instead of choosing points at random, for each value of x, 100 values of y at fixed intervals were generated. This was done for 100 x values, also chosen at fixed intervals. These points were input to the network and the network outputs calculated. Network outputs belonging to class $(1, 0)$ are shown in black and network outputs belonging to class $(0, 1)$ are shown in gray. The resolution of the points between 0 and 1 was large enough to show the decision boundary between the two classes. For a training set with no noise the decision boundaries look very similar.

The same data set containing no added noise was used again to train a network

Cross-Entropy	NDL, $\epsilon = 0.05$	NDL, $\epsilon = 0.03$	NDL, $\epsilon = 0.01$
96.25	96.15	96.20	96.1

Table 5.1: The maximum correct classification achieved for a series of networks in a training α-survey for a data set with no added noise.

Cross-Entropy	NDL, $\epsilon = 0.05$	NDL, $\epsilon = 0.03$	NDL, $\epsilon = 0.02$	NDL, $\epsilon = 0.01$
85.05	84.05	83.25	82.35	81.05

Table 5.2: The maximum correct classification achieved for a series of networks in a training α-survey for a data set with 10% miss-classified data.

with the NDL cost function for two smaller values of ϵ. These showed similar results to those in Figure 5.2. The maximum percentage of correct classifications in each case of training is shown in Table 5.1. The results were similar for each network.

The two methods of training were compared with a training set that contained 10% miss-classified data, $\phi = 0.1$ and $\sigma = 0.0$. Figure 5.4 and 5.5 show the results for training with the Cross-Entropy and the NDL cost function with different values of ϵ. The second derivative of the Cross-Entropy α-survey in Figure 5.4e does not point to the maximum correct classification, but is quite close. For each value of ϵ used in NDL training the second derivative does point to the maximum of correct classifications except when training was performed with $\epsilon = 0.02$ in Figure 5.5b. Table 5.2 shows the maximum of correct classifications made for each of the networks for the test set. The largest percentage of correct classifications on the test set was achieved by the network trained using the Cross-Entropy cost function. As the value of ϵ decreased in NDL training the performance of the networks on the test set grew worse, but with a difference of approximately 4% in correct classifications.

Figure 5.6 shows the results of training with a data set where the two classes are allowed to overlap each other along the boundary between the two classes. This arrangement of points was used instead of having random points that are miss-classified.

Cross-Entropy	NDL, $\epsilon = 0.05$	NDL, $\epsilon = 0.02$
91.1	90.3	89.85

Table 5.3: The maximum correct classification achieved for a series of networks in a training α-survey for a data set with overlapping classes.

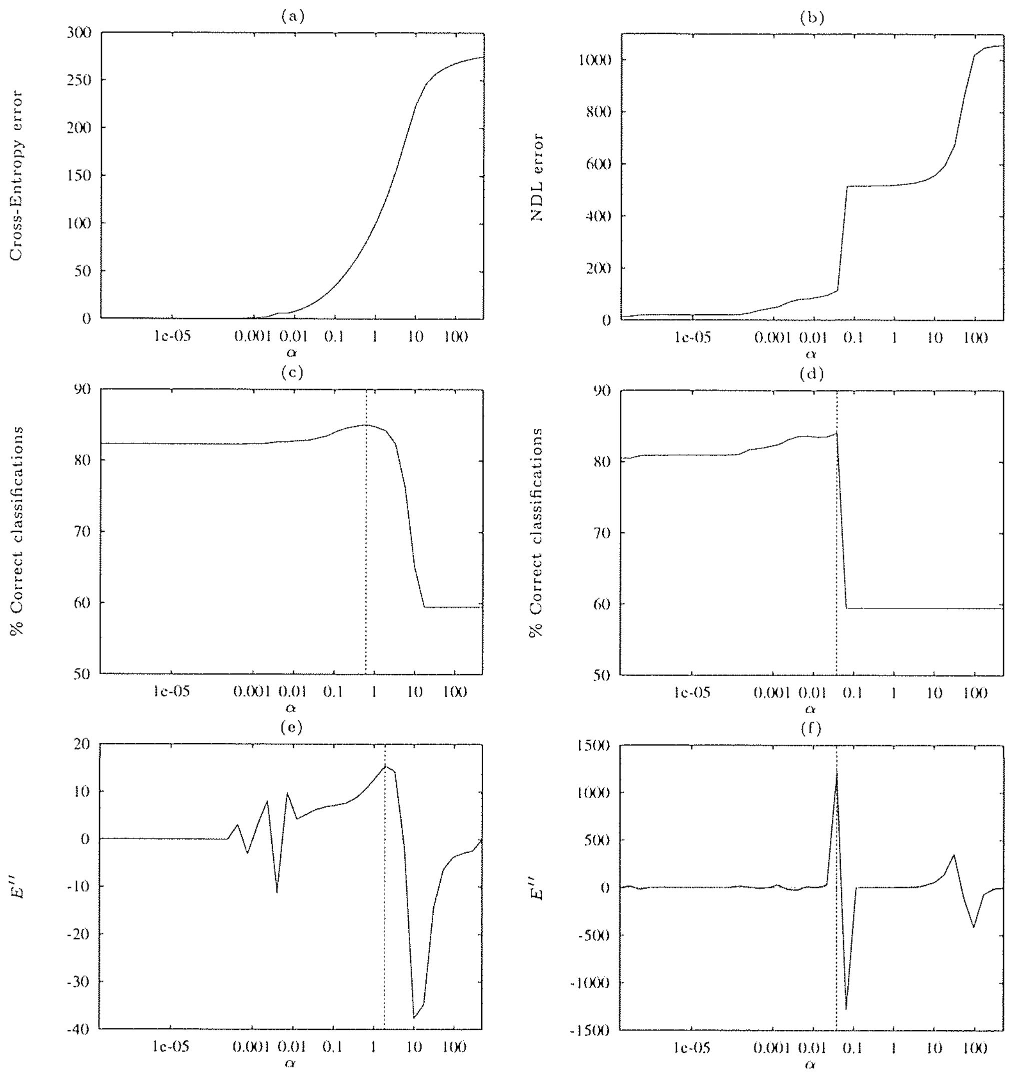

Figure 5.4: The Cross-Entropy and NDL data error for training and testing for the two-class data set with 10% miss-classified data. The vertical lines indicate the maximum percentage of correct classifications achieved on the test set and the maximum second derivative. (a) The Cross-Entropy data error for training α-survey. (b) The NDL data error for training α-survey. (c) The percentage of correct classifications on the test set, for a network trained with the Cross-Entropy cost function. (d) The percentage of correct classifications on the test set, for a network trained with the NDL cost function, with $\epsilon = 0.05$. (e) The second derivative of the training NDL data error. (f) The second derivative of the training Cross-Entropy error.

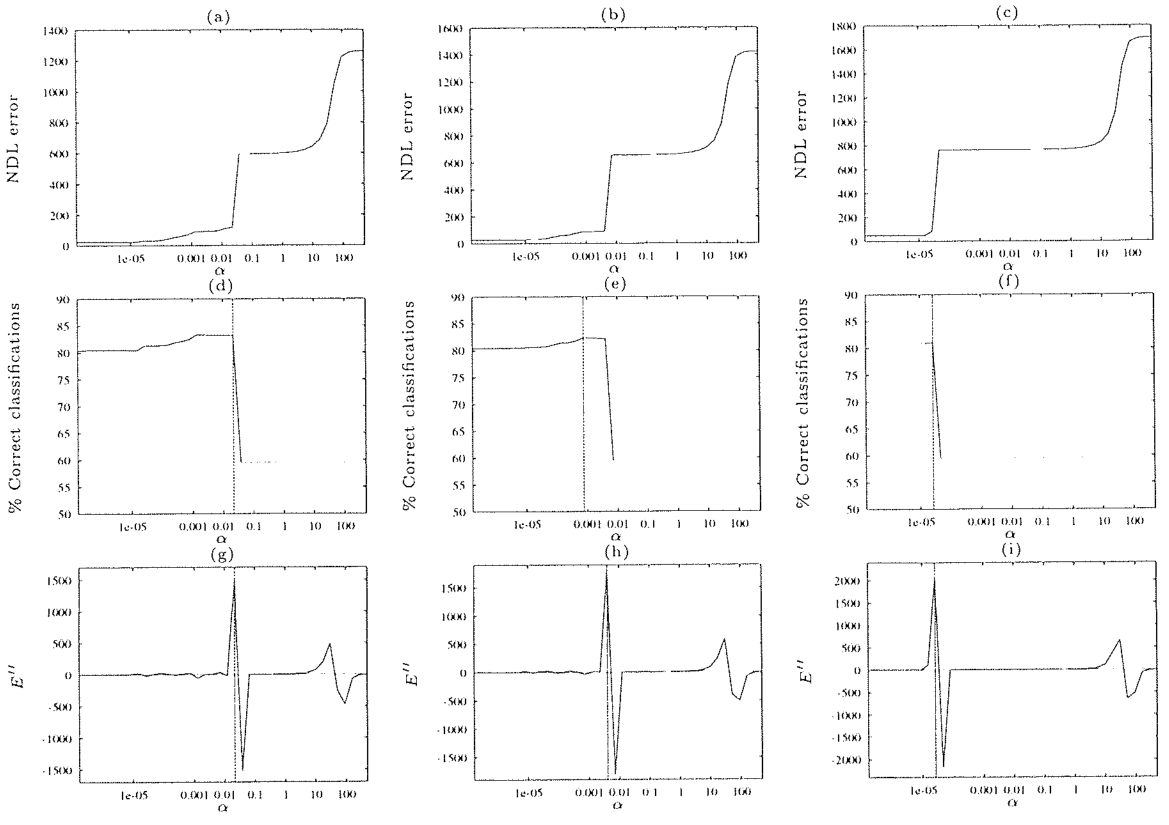

Figure 5.5: The NDL data error for training and testing for the two-class data set with 10% miss-classified data. (a) The NDL data error for training α-survey with $\epsilon = 0.03$. (b) The NDL data error for training α-survey with $\epsilon = 0.02$. (c) The NDL data error for training α-survey with $\epsilon = 0.01$. (d) The percentage of correct classifications on the test set, for a network trained with the NDL cost function. (d), (e), (f) The percentage of correct classifications on the test set, for a network trained with the NDL cost function and different ϵ. (g), (h), (i) The second derivative of the training NDL data error for different ϵ.

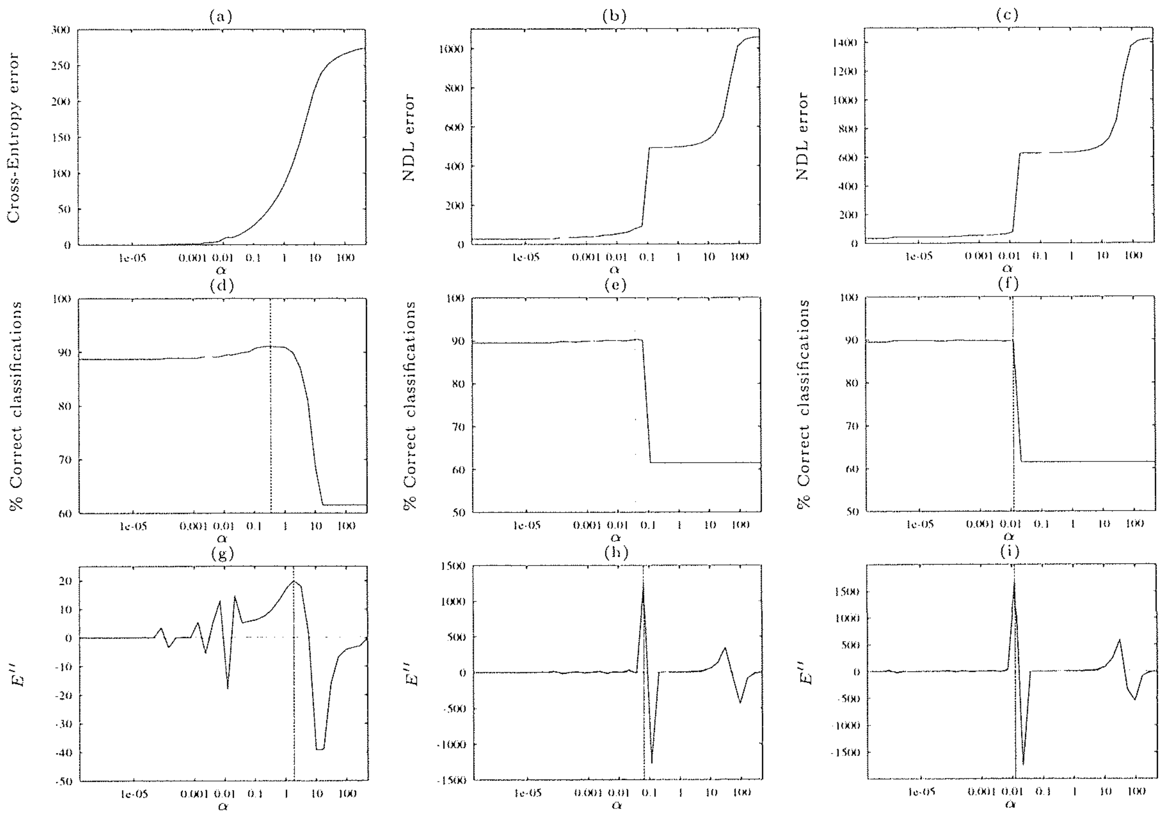

Figure 5.6: The Cross-Entropy and NDL data error for training and testing for the overlapping classes data set. (a) The Cross-Entropy data error for training α-survey. (b) The NDL data error for training α-survey, with $\epsilon = 0.05$. (c) The NDL data error for training α-survey, with $\epsilon = 0.02$. (d) The percentage of correct classifications on the test set, for a network trained with the Cross-Entropy cost function. (e), (f) The percentage of correct classifications on the test set, for a network trained with the NDL cost function for different ϵ. (g) The second derivative of the training Cross-Entropy error. (h), (i) The second derivative of the training NDL data error for different ϵ.

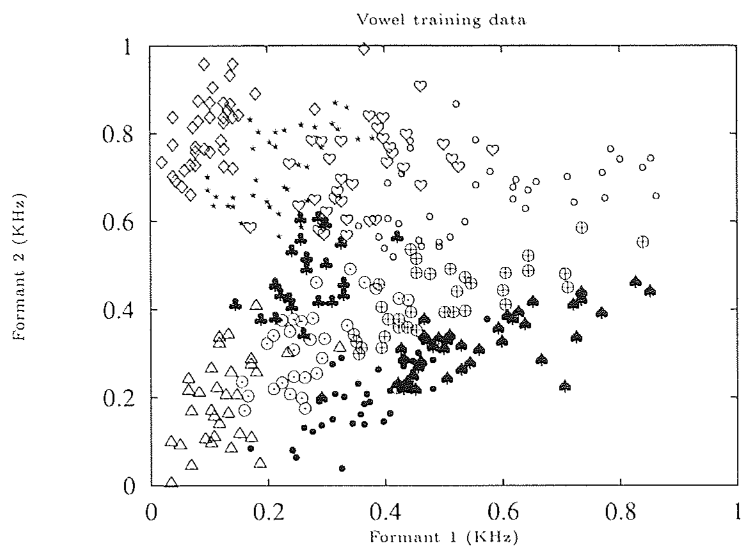

Figure 5.7: The Vowel training data showing the first and second formant frequencies plotted against each other. The labels have the following Arpabet notation: $\heartsuit$ = e, $\star$ = I, ♠ = a, $\circ$ = @, $\bullet$ = c, ♣ = R, $\diamondsuit$ = i, $\oplus$ = A, $\triangle$ = u, $\odot$ = U.

For the data set used here the two noise parameters were $\phi = 0.0$ and $\sigma = 0.02$. The results are similar to those presented earlier. The second derivative of the training α-survey points to the maximum correct classifications for the series of networks in the α-survey for $\epsilon = 0.02$. Table 5.3 shows again that as the value of ϵ decreases the maximum correct classifications decreases. The difference in correct classifications of approximately 1% is not significantly worse, with NDL training with $\epsilon = 0.02$, than with Cross-Entropy training.

5.2 Using NDL on real-life classification problems

The NDL cost function was used with some real life data sets. Two of these data sets were used in the European Community ESPRIT project 5170, the Statlog project, and are the Heart disease data set and the Diabetes data set. Details of these data sets can be found in Michie *et al.* [52]. Another data set which is used for benchmarking purposes is the Vowel data set collected by Peterson and Barney [61]. For each problem, the data was split into a training and test set. The choice of value for the regularising parameter α indicated by the training α-survey could then be checked against actual performance for a test set.

5.2.1 Vowel data

A problem that contains many more classes is the Vowel recognition data set shown in Figure 5.7. The data shows the digitised plot of the first and second formant frequencies for men, women and children for 10 vowels. This data set has been used by Huang and Lippmann [32] for the comparison of Neural Networks with conventional classifiers.

Figure 5.8 shows the results of training with the Cross-Entropy and NDL cost function where $\epsilon = 0.05$. The second derivative of the training α-survey shows a peak for Cross-Entropy training that does not point to the maximum correct classification of 81.38%. The NDL training α-survey shows a small 'cliff' at the base of the main 'cliff' in Figure 5.8b. The second derivative of the training α-survey shows two peaks corresponding to these steps. The larger peak points to a value of α that is very close to the value of α used in a network that gave the maximum correct classification of again 81.38%.

Figure 5.9 shows the results of testing the network with a set of points designed to show the boundaries between the classes. The networks gave the same performance on the test set but give very different decision boundaries. In Figure 5.9b some of the classes have been ignored in the network outputs.

The maximum percentage of correct classifications for the test set decreased as the value of ϵ was decreased as shown in Figure 5.10. The maximum correct classification was 80.48% for NDL training with $\epsilon = 0.03$ and 79.88% for NDL training with $\epsilon = 0.01$. The second derivative of the training α-survey in Figure 5.10 did not point to the maximum correct classification, neither did the second largest peak. Results for NDL training with $\epsilon = 0.01$ were slightly better but show that varying ϵ in training can lead to very different results.

The decision boundaries for a network trained with the NDL cost function with $\epsilon = 0.03$ is shown in Figure 5.11. These boundaries look different to the ones when NDL training with $\epsilon = 0.05$ was used. Again this shows that different values for ϵ in training can give different results. A smaller value of ϵ gave a smaller maximum correct classification for the test set.

5.2.2 Heart Desease data set

The Heart disease data set originally comes from the Cleveland Clinic Foundation. It was also part of the Statlog project [52], which provided a comparison of many different approaches to classification. The Diabetes data set was also one of the test problems used in this project. The Heart disease data set contains data on the presence or absence of heart disease in a patient. For this data set there are 13 attributes taken from results of a series of medical tests. This data set also has a cost matrix but this is

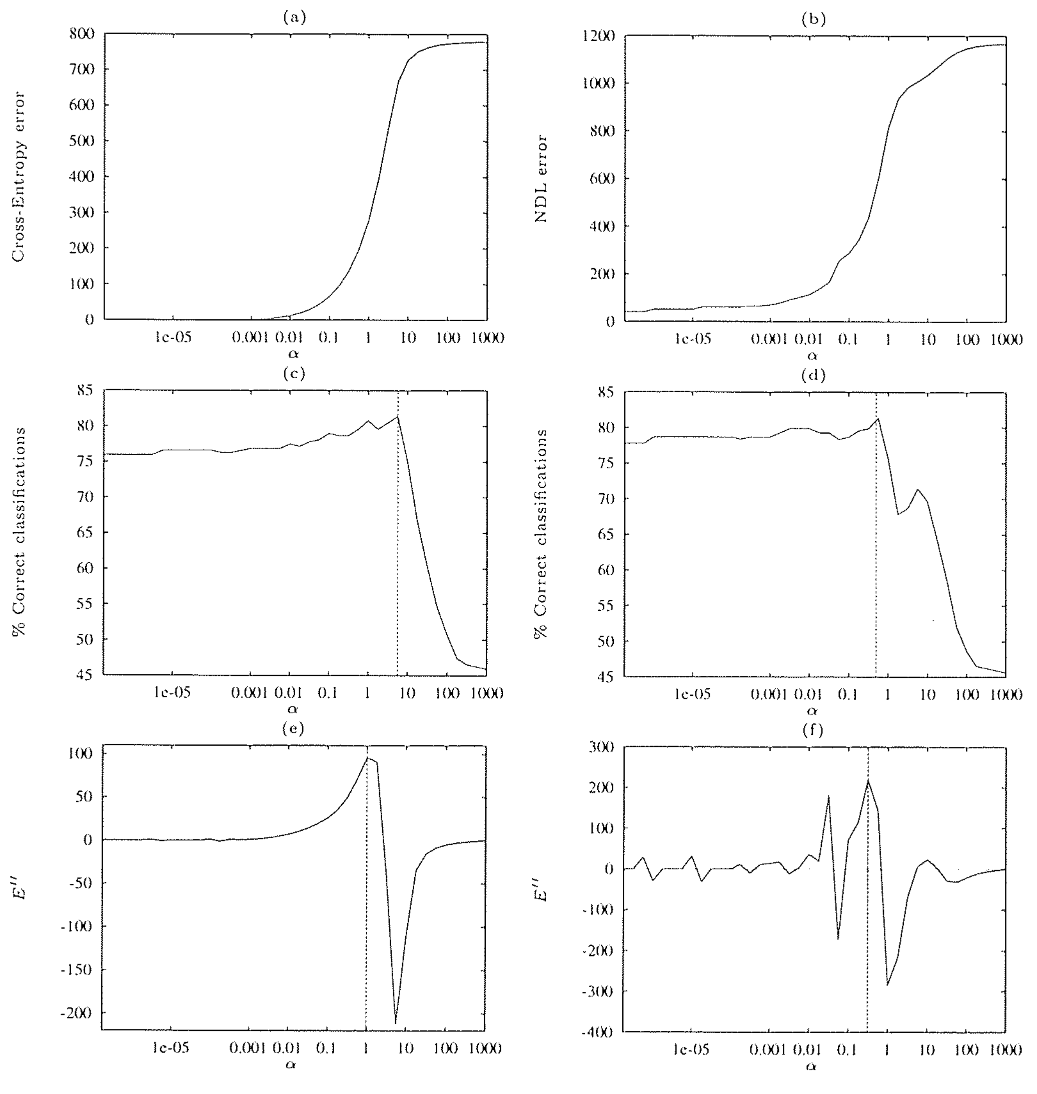

Figure 5.8: The results of training with Cross-Entropy and NDL cost function for the Vowel data set. The vertical lines indicate the maximum percentage of correct classifications achieved on the test set and the maximum second derivative. (a), (c), (e) The training α-survey, the percentage of correct classifications made on the test set and the second derivative of the training Cross-Entropy data error. (b), (d), (f) The training α-survey using the Cross-Entropy cost function, the percentage of correct classifications made on the test set and the second derivative of the training NDL data error.

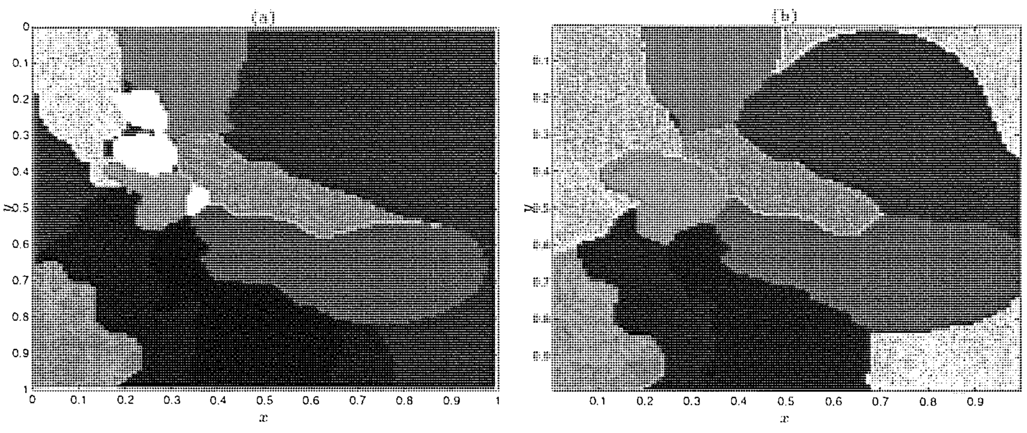

Figure 5.9: Decision boundaries for the different classes in the Vowel data set. Each shade of gray corresponds to a different class. (a) A network trained using the Cross-Entropy cost function. (b) A network trained using the NDL cost function with $\epsilon = 0.05$.

not considered here. An RBF network using the NDL cost function was trained with a number of basis functions equal to the number of patterns in the training set. The number of patterns was 200. The test set contained 70 patterns. Figure 5.12 shows the results for training with two values of ϵ. For both networks the second derivative of the training α-survey showed single large peaks. For the network trained with $\epsilon = 0.05$ the second derivative failed to point to the maximum test set correct classifications, the maximum being 81.43%. Training with a slightly smaller value of ϵ did cause the second derivative of the training α-survey to point to the maximum correct classifications. However the maximum correct classifications achieved was smaller at 80.00%.

5.2.3 Diabetes data set

The Diabetes data set records the positive tests for diabetes according to World Health Organisation criteria. These criteria are based on physiological and medical test results. The data set contains 8 attributes based on these test results. More details can be found from the results of the Statlog project [52]. RBF networks were trained with the NDL cost function with two values of ϵ. Again the number of basis functions was equal to the number of patterns in the training set. The training set contained 300 patterns and the test set contained 468 patterns. Figure 5.13 shows the results of training with $\epsilon = 0.05$ and $\epsilon = 0.03$. The second derivative of the training α-survey for a network trained with $\epsilon = 0.05$ showed one large peak and one small peak. The small peak pointed to the maximum correct classifications correct in this case. Training with a smaller value of ϵ caused a peak in the second derivative that did indicate a good value

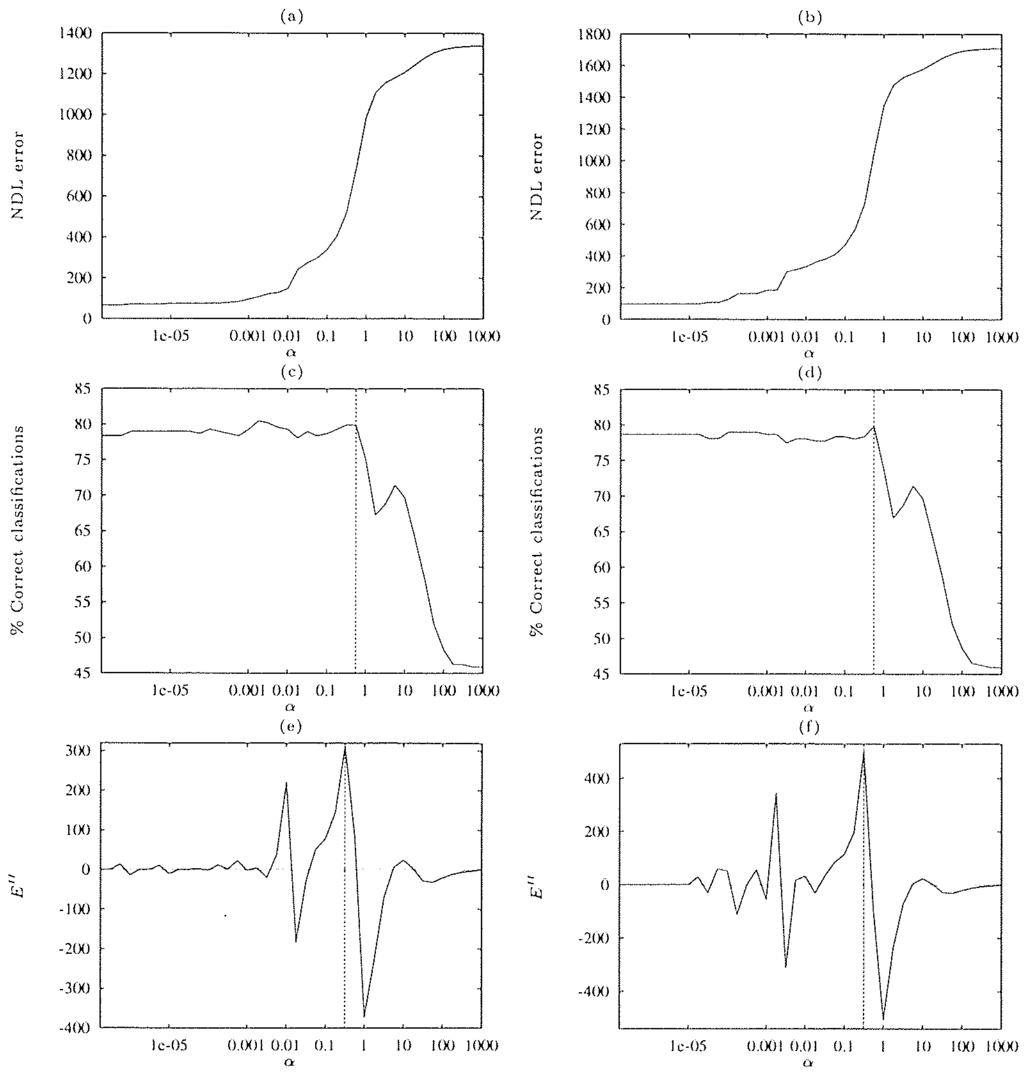

Figure 5.10: The results of training with the NDL cost function on the Vowel data set using different values for ϵ. (a), (c), (e) The training α-survey, the percentage of correct classifications made on the test set and the second derivative of the training NDL data error with $\epsilon = 0.03$. (b), (d), (f) The training α-survey, the percentage of correct classifications made on the test set and the second derivative of the training NDL data error with $\epsilon = 0.01$.

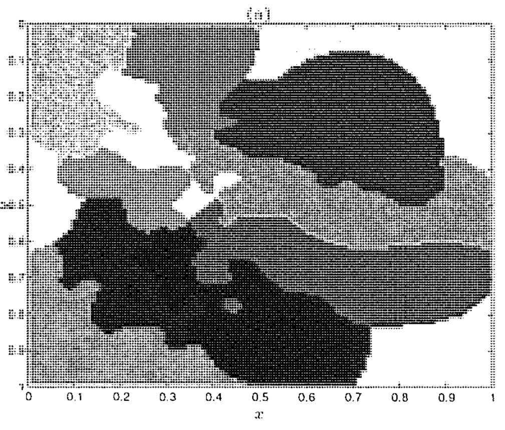

Figure 5.11: Decision boundaries for the different classes in the Vowel data set for a network trained with the NDL cost function, $\epsilon = 0.03$.

for α for the test set. The maximum correct classifications was again smaller with a smaller value of ϵ.

5.3 Summary

After examining the behaviour of α-surveys in regression problems, the same procedure was applied to classification problems. This time the peaks in the α-surveys would point to the maximum classification rate. A simple two-class problem was presented with clearly separated clusters. The peaks in the second derivative calculations for NDL cost function training did point to the maximum classifications correct in this case. This was compared to the Cross-entropy cost function whose second derivative peaks did not point to the maximum classifications correct. Two techniques of adding random noise were used and the same α-surveys generated. Results for NDL training, though still pointing to the correct positions, gave generally worse classifications correct compared to Cross-Entropy with the same data sets. This effect could not be explained. Several real-life data sets were used and results were good with the Heart disease data set. Results were very poor for the Diabetes data set. Minor differences occurred in the Vowel data, which gave good results for larger values of ϵ. Results were poor with smaller values of ϵ.

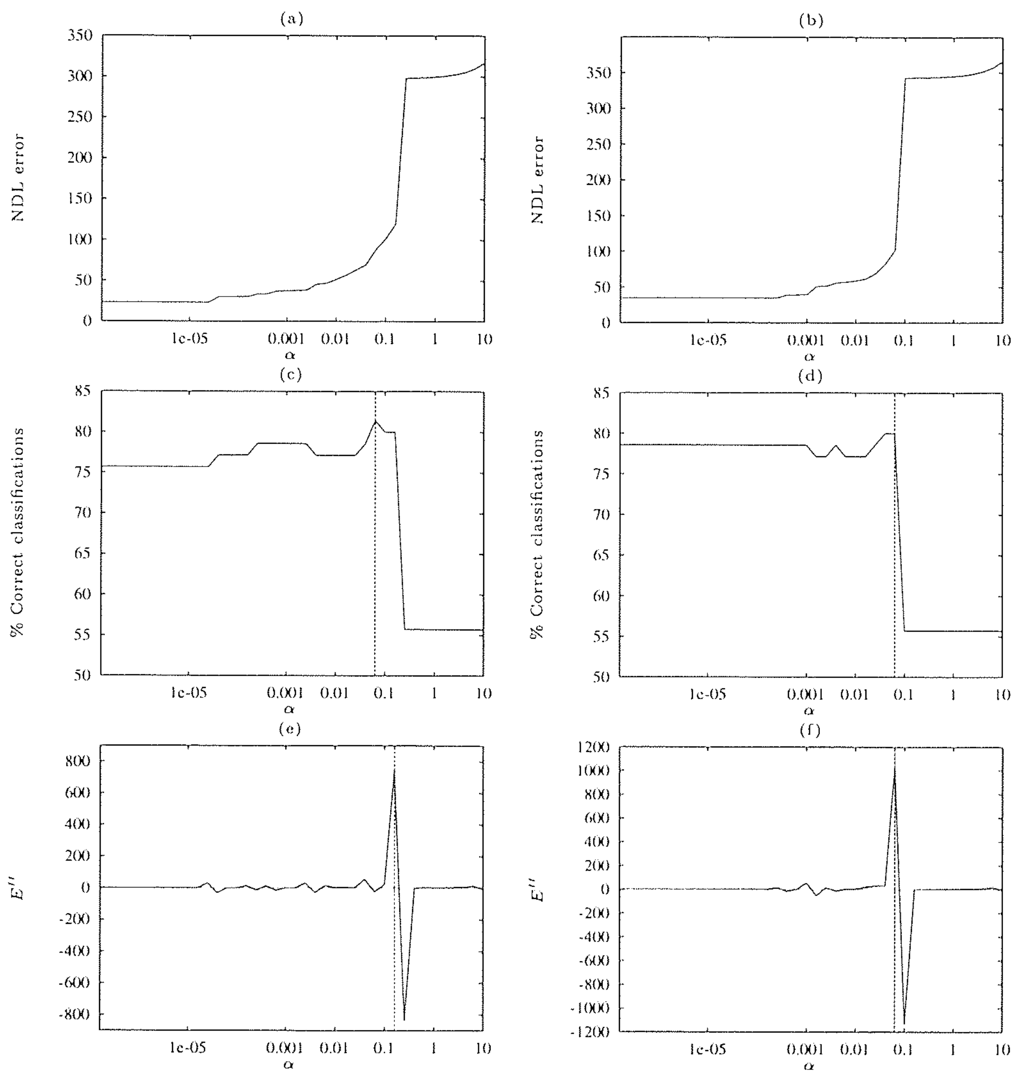

Figure 5.12: The results of training with the NDL cost function on the Heart disease data set using different values for ϵ. The vertical lines indicate the maximum percentage of correct classifications achieved on the test set and the maximum second derivative. (a), (c), (e) The training α-survey, the percentage of correct classifications made on the test set and the second derivative of the training NDL data error with $\epsilon = 0.05$. (b), (d), (f) The training α-survey, the percentage of correct classifications made on the test set and the second derivative of the training NDL data error with $\epsilon = 0.03$.

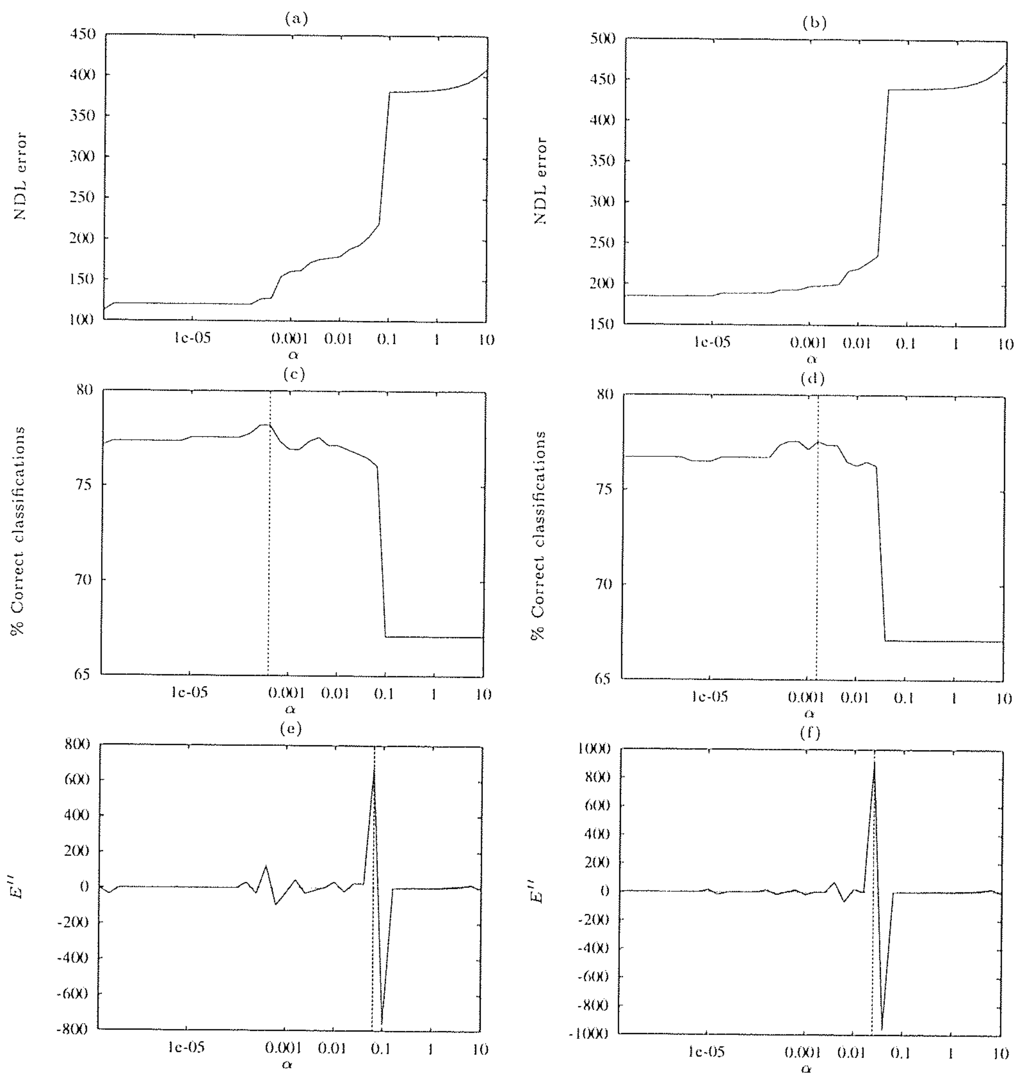

Figure 5.13: The results of training with the NDL cost function on the Diabetes data set using different values for ϵ. (a), (c), (e) The training α-survey, the percentage of correct classifications made on the test set and the second derivative of the training NDL data error with $\epsilon = 0.05$. (b), (d), (f) The training α-survey, the percentage of correct classifications made on the test set and the second derivative of the training NDL data error with $\epsilon = 0.03$.

Chapter 6

NDL and time series prediction

6.1 Time series prediction and Neural Networks

Multi-valued mappings can also occur in time series prediction problems. Again a network that converges to a mode may be desired instead of one that converges to the mean. In Section 3.1.1 an 'inverse' XOR time series problem was presented where a deterministic series was corrupted by random noise. In this experiment a network was trained on a single input to predict the target. In this section problems that require the use of many time-steps will be considered. The 'inverse' XOR problem showed that a network was needed to model a likely target instead of the mean of the targets and to model that target accurately. In this particular problem the different multi-valued targets for a given input were equally likely. The accuracy can be measured using the ϵ parameter in the NDL cost function. For other time series problems there may be a number of deterministic paths present in the data. With NDL cost function training the network will give outputs that will follow more 'closely' the outputs of the generator of the data. The network outputs are then more likely to stay correct for longer. However when the next time-step in a series does not follow the most likely path the error will be greater.

6.1.1 The details of the time series experiments

Before describing the individual experiments carried out the methods used for dealing with time series prediction are discussed. This section describes how a network is trained from a time series data set in one-step-ahead mode. The process of calculating the mean data error and its error-bars are discussed for iterated prediction.

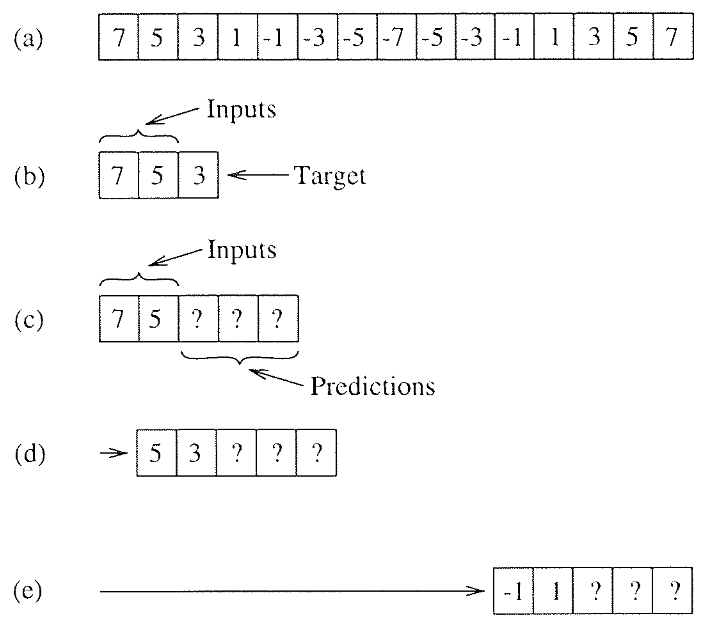

Figure 6.1: How training and testing is performed on time series prediction problems. (a) An example time series where each time-step contains a single observation. (b) How a number of time-steps can be fed into a network to train on a single target. (c) The inputs are copied from the original time series and a set of predictions are made. (d),(e) Starting from a different place in the time series different iterated predictions can be made.

Time series training

Most of the time-series prediction data sets presented in this section contain a number of different observations at different times. Each observation contained a number of attributes or dimensions. For example some problems had single valued inputs making a vector of time series observations, $x_{t-r}, x_{t-r+1}, ..., x_t$. In time series prediction tasks the targets are part of the series. If training is performed on a single time-step, this means that the current observation x_t is used as the input and the next time-step is used as the target x_{t+1}. Another arrangement is to have more than one time-step in the input to the network. This will increase the number of inputs in the input layer. In this case the input pattern $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ will be used. For all arrangements of time-steps a single target or prediction was used.

Prediction sequences

For the experiments presented in this chapter it was required that iterated prediction would be performed. Once a network was trained using one-step-ahead prediction, the networks were tested with iterated prediction. With this type of prediction it would be expected that the predicted error would become compounded. This happens as small prediction errors at the beginning of the iterated prediction sequence lead to large errors at further time-steps in the same sequence. If a single time-step was used in training then the first test set pattern would start the iterated sequence. If two time-steps were used in training then the first two time-steps in the test set would start the iterated prediction. The first inputs of the iterated prediction are input to the network and a prediction of the target x_{t+1} is calculated. This prediction then becomes part of the input to finding the prediction of the target x_{t+2}. This continues until a desired number of predictions are made into the future. For the experiments in this chapter 20 to 30 iterated predictions made up a sequence. This is a single prediction sequence from the first test set inputs. Another iterated prediction sequence could be generated from the second test set inputs. The number of time-steps affects the number of patterns that can be presented to the network. The number of iterated predictions in an individual sequence also affects the total number of iterated prediction sequences that can be made. Figure 6.1 shows an example time series with single attribute observations making up the series and how the iterated predictions are made from different starting points. In this way the results from a set of iterated prediction sequences can be used to test how accurately it followed the actual data in the test set. For each iterated prediction the NDL or sum-of-squares cost function can be used to calculate an error between the iterated prediction and the actual data. The error for each iterated prediction for the whole set of iterated prediction sequences can be used to calculate the mean and variance for each time-step in the set of iterated prediction sequences. The mean and variances can then be used to plot the mean and error-bars.

A single iterated prediction sequence

In order for the test set mean NDL data error to be calculated the outputs for every iterated prediction sequence have to be calculated. Some of the graphs presented in this chapter show individual iterated prediction sequences. In these graphs the iterated sequence of outputs given by the network are shown against the actual sequence that occurred in the test set. Different individual iterated prediction sequences will show very different results depending on the initial starting point from the set of possible starting points. When the number of dimensions in the inputs is greater than 3, individual iterated prediction sequences become more difficult to visualise. In the following experiments, individual iterated prediction sequences were only shown for time series prediction problems where the number of inputs did not exceed 3.

Error time-steps

Some of the graphs shown in this chapter plot the error at two time-steps within an iterated prediction sequence for all sequences. For example the error at 4 time-steps into a prediction sequence for the set of sequences beginning at time τ may be recorded. This could be compared to the error at 5 time-steps into the same prediction sequence for the same set of sequences. This shows how the iterated prediction error at two time-steps are related. If the network makes predictions that follow the most likely path for the data then the error for two consecutive time-steps may both be small. If the network makes a wrong prediction then the error for two consecutive time-steps may be large. This will give some indication of the confidence to be placed in some predictions. For the experiments presented here only consecutive time-step errors were used. In most cases the error at the first and second time-steps were plotted against each other. In a small number of other tests consecutive time-step errors were plotted from the middle of the iterated prediction sequences.

6.2 Using NDL in time series prediction

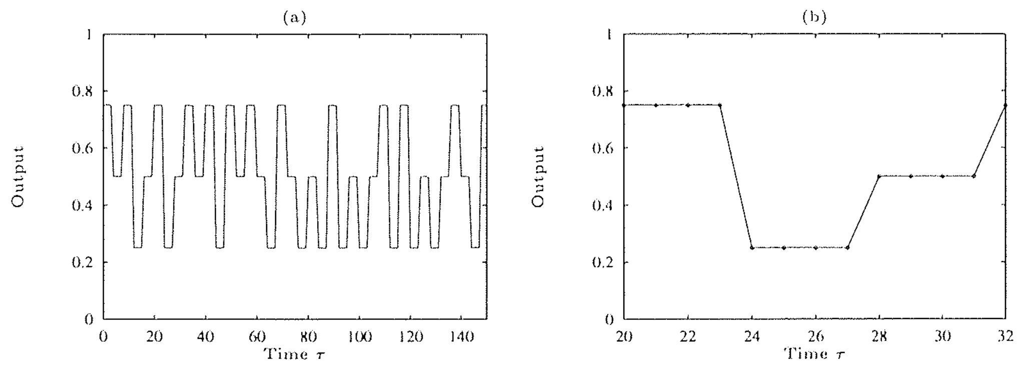

Figure 6.2: A data set showing a regular time series prediction problem. (a) The training data. (b) Part of the training set showing the transitions in the series.

Figure 6.2a shows a data set created to show the prescence of multi-modal mappings in a time series. A data generator is presumed that produces a sequence of numbers at three values, the values being 0.25, 0.5 and 0.75. The function produces 4 outputs of each value consecutively. The probability of a transition to any of the other values is equal. For example 4 values of 0.5 may be followed by 4 values of 0.75 or 4 values of 0.25 with equal probability. Figure 6.2b shows a small range of values in the data set to show the transitions more clearly. This data set was used to show the deterministic and random characteristics in some data. Once the data is found to be at a particular

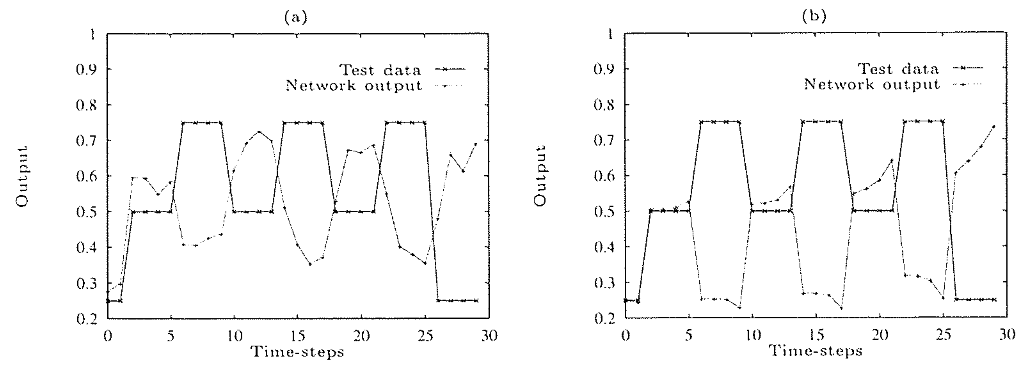

Figure 6.3: Iterated prediction sequences for the tri-valued time series test set. Sequences from time τ in the test set are compared to the actual data points in the test set. (a) A network trained with the sum-of-squares cost function. (b) A network trained with the NDL cost function.

value the next 3 data set values are completely deterministic. However the next value is completely random. The network was presented with 200 values from this sequence. Another 200 values were generated as a test set.

This data was used to train a single hidden layer MLP with 10 hidden units using the NDL cost function. The value of ϵ was 0.03 and the number of time-steps was 5. The number of time-steps was chosen so that it would capture the possible transitions from one value to another. This many time-steps would make sure that the network would know when the transitions in the data occur. An investigation into how the number of time-steps affects test results is presented later in this chapter. At the same time a network was trained with the sum-of-squares cost function and tested with the NDL cost function. An α-survey was not generated for this training set. Therefore training was carried out with no regularisation.

Figure 6.3a shows the iterated prediction output of the network trained using the sum-of-squares cost function and the actual time series data. Iterated prediction is shown from time $\tau = 0$ in the test set. The iterated sequence was generated by taking the first 5 values of data in the test set, from $\tau = 0$, and generating the next 30 iterated predictions from these initial inputs. Training the network using this cost function has failed to capture the multi-valued transitions. For example at time $\tau + 3$ the network has output the average of the two possible transition values of 0.5 and 0.75. The output from the iterated sequence then degenerates quickly and the structure of the input has not been retained in the predictions. The network has failed to recognise that there are three distinct values produced by the time series data generator.

Figure 6.3b shows the same iterated prediction output sequence as shown in 6.3a where the network has been trained using the NDL cost function. The predicted

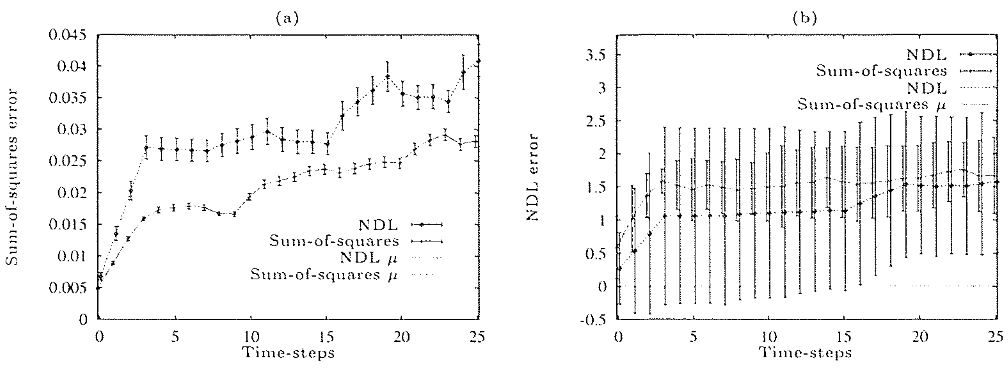

Figure 6.4: The results of testing two networks trained using the NDL and sum-of-squares cost function on a tri-valued time series data set. (a) The mean sum-of-squares data error and its error-bars for iterated prediction on a test set for both NDL and sum-of-squares cost function trained networks. (b) The mean NDL data error and its error-bars for iterated prediction on a test set for both NDL and sum-of-squares cost function trained networks.

output looks more like the original time series pattern. Even though the predictions were wrong from time-step 6 to 9, they are still predictions that the data generator could output. The predictions have a chance of following a path accurately after several time-steps where its predictions are wrong. The probability that the predictions are correct depend on the number of different paths the data generator can output. For example the transition from time-step 9 to time-step 10 has resulted in the network producing a predicted output very close to the original data even though the last four predictions were wrong.

The network produces outputs that follow more closely the valid outputs of the time series data generator. Again the type of transition that the network converges on will be the same and will depend on the frequency of any particular transition. For example if transitions from 0.5 to 0.25 are more common than from 0.25 to 0.75 the network trained with the NDL cost function will converge to a 0.5 to 0.25 transition. NDL cost function training in this case has chosen 0.25 as the value following 4 values of 0.5 and 0.5 after 4 values of 0.25. Transitions to 0.75 have been ignored. This may be an undesirable feature in that the iterated prediction sequence follows a subset of possible values.

Figure 6.4 shows the mean NDL data error and its error-bars for iterated prediction on a test set. These two graphs compare performance of a network trained using the sum-of-squares cost function against training using the NDL cost function. For all the experiments in this chapter the networks trained using the sum-of-squares and NDL cost function were tested with the NDL cost function. If the two networks were both tested with the sum-of-squares cost function we would expect the mean sum-of-squares

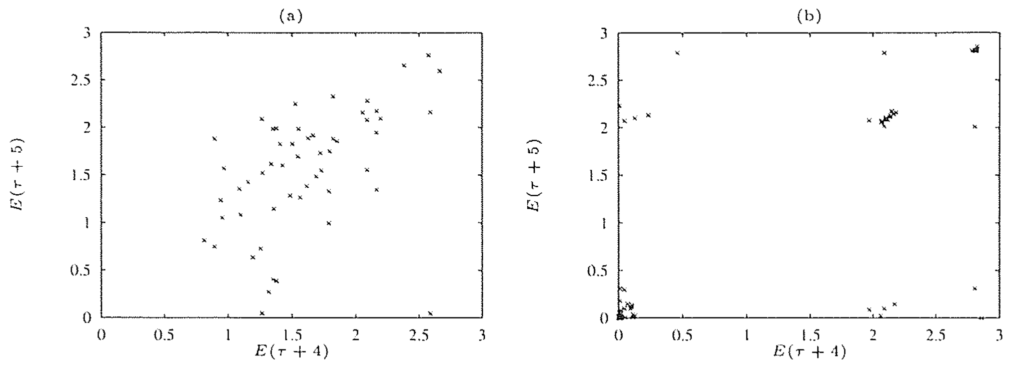

Figure 6.5: The NDL data error at different time-steps in the set of iterated prediction sequences for the tri-valued time series test set. (a) A network trained with the sum-of-squares cost function. (b) A network trained with the NDL cost function.

data error for both networks to be like that shown in Figure 6.4a. The mean sum-of-squares error for an NDL cost function trained network is higher because of the possibility of wrong predictions. The variance of the error will also be greater. The two networks cannot be compared using the sum-of-squares error. This is because the cost function does not measure how close the predicted output was to a more probable output regardless of whether it occurred or not. Figure 6.4b shows that the mean NDL data error for NDL training is smaller than the sum-of-squares trained network. It shows that the predictions are more accurate when the network outputs follow the correct path. The error-bars for the NDL trained network are wider because of the greater error in wrong predictions.

Figure 6.5a shows the NDL data error from the network at time $\tau + 4$, where τ is the starting time-step within the test set, against the NDL data error at time $\tau + 5$. This graph shows that there are no clusters in the NDL data error at one time-step and the next using sum-of-squares as the training cost function. However we can see from the original data that if we could accurately predict a transition to a sequence of values then the NDL data error for the following time-steps should be small. Figure 6.5b shows that the NDL data error at two time-steps forms a number of clusters. There is a large cluster of points around the origin, showing that for a large number of predictions the error for the next time-step will be small also. Therefore if we have calculated the error for the current time-step and it is small, it is likely that the next NDL error will also be small. This gives the network the ability to give some indication of the accuracy of its predictions. Large errors may occur when a transition occurs, but this is the non-deterministic part of the time series. It would not be expected that the network give correct predictions all the time. This graph also shows when bad predictions are likely to occur. Figure 6.5b shows that if the error

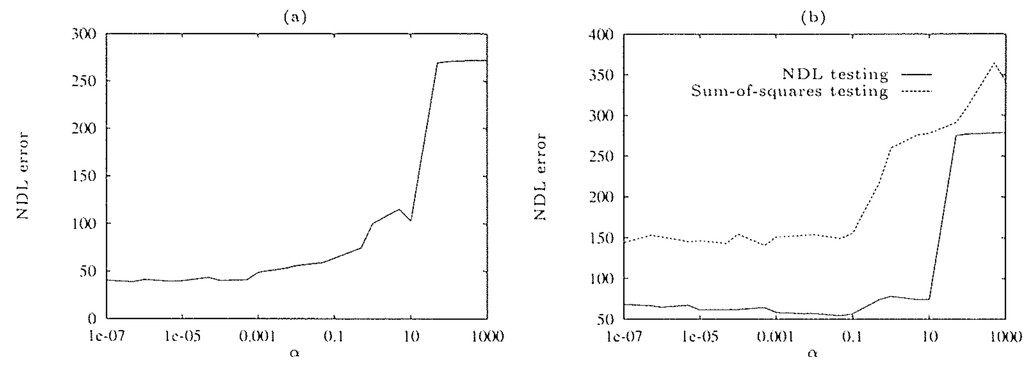

Figure 6.6: The plots show results from an α-survey for the tri-valued time series data set. (a) The training NDL data error. (b) The test set NDL data error for networks trained with the NDL and sum-of-squares cost function.

is large then the next time-step is likely to be large also. Therefore if the network predicted an output with a large error to the actual predicted output then it would be wise not to accept the network's prediction for the next time-step. This clearly shows that there are times when to accept the prediction of the network and times when the prediction should not be accepted. When the sum-of-squares cost function was used in training it gave no indication of the confidence to be placed in the next prediction.

6.2.1 Generating an α-survey on the time series

In Section 6.1.1 it was shown that by training a network over a range of values for the regularisation parameter α, this would give some insight into the levels of structure in the data. This was shown by 'cliffs', when α was plotted against the data error. The same experiment was carried out on this artificial time series data set with some added noise. A single hidden layer MLP was used using linear output units. The number of hidden units was 20 and the inputs were coarse-coded with 4 Gaussians. The network was trained with 5 time-steps.

Figure 6.6 shows training and testing for networks trained with a range of values for the regularising parameter α. The noise in the data took the form of Gaussian noise with a fixed variance of 0.03 with the centre taken as the value of the original input. This was done to see the effect of network regularisation on a noisy time series that contains multi-modal transitions. In Figure 6.6a the NDL data error over the training set decreases sharply as α is reduced over a small range until α is around 10. At large values of α the network model will give the same output for all inputs. At small values of α the network follows one of the different transitions. At each value of α the network was tested and the test NDL data error shows a minimum between

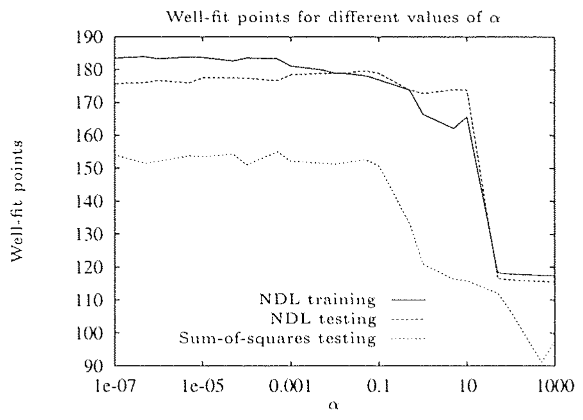

Figure 6.7: The number of well-fit points for the α survey on the tri-valued time series data set.

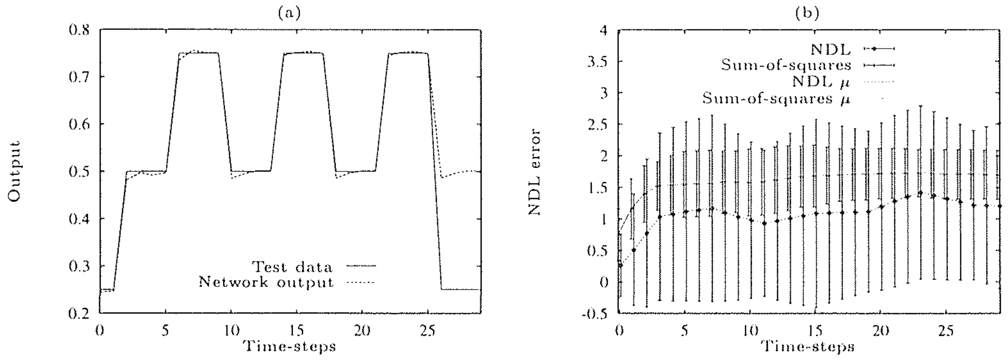

Figure 6.8: The results from an α-survey on a noisy tri-valued time series training set. (a) The iterated prediction sequence from time τ in the test set of a network trained with $\alpha = 0.05$. (b) The NDL data error mean and error-bars for the set of iterated prediction sequences.

overfitting and underfitting at an α value of about 0.05. At small values of α, training has resulted in a network that has overfitted the data. After α has decreased to the value of 0.05 the test set NDL data error increases again. The NDL data error for the network trained using the sum-of-squares cost function was greater for all values of α in Figure 6.6b. Figure 6.7 shows how the number of well-fit points suddenly increases as the value of α is decreased to a certain point. The maximum number of points fit by the model for the test set corresponds to the minimum NDL data error achieved for the test set. Networks trained using the sum-of-squares cost function were tested using the NDL cost function and the number of well-fit points calculated. This method of training did not fit as many points 'closely' as the NDL training method.

Figure 6.8a shows the iterated prediction sequence for the network trained with

t = 1		t = 4				
Input	Target	Input				Target
.5	.5	.25/.75	.25/.75	.25/.75	.5	.5
.5	.5	.25/.75	.25/.75	.5	.5	.5
.5	.5	.25/.75	.5	.5	.5	.5
.5	.25/.75	.5	.5	.5	.5	.25/.75

Table 6.1: The inputs for different time-steps and their targets, for a network using 1 time-step and another using 4 time-steps. The symbol '/' separates the alternative input values.

$\alpha = 0.05$. This was the network giving the best performance or minimum test set NDL data error. The iterated sequence has coincidentally picked out many correct transitions. Performing the same iterated prediction from a different starting point in the test set may have given many incorrect transitions. The mean NDL data error and its error-bars in Figure 6.8b show that the mean NDL data error is smaller than when the sum-of-squares cost function is used. The error-bars for the NDL trained network are also wider than those for the sum-of-squares trained network. This is because the NDL trained network will in some cases predict an incorrect output. In these cases the NDL data error will be larger than for a network trained with the sum-of-squares cost function, which predicts an average output. It is also shown in Figure 6.8b that the error-bars for an NDL trained network do not extend too much beyond the error-bars for the sum-of-squares trained network.

6.3 Time-steps and prediction error

The number of time-steps used in these experiments was important. In the example of the tri-valued series the number of time-steps used in the input to the network was 5. This was chosen deliberately to capture when a transition would occur from one value to another. Table 6.1 shows how the number of time-steps used in the input will affect the outcome of training. It shows the possible transitions from 0.5 to either 0.25 or 0.75. When the number of time-steps is 1 there are 3 instances where a value of 0.5 changes to 0.5 in the next time-step and one instance where 0.5 changes to 0.25 or 0.75. Training using the NDL cost function for each of these input, target pairs will converge on 0.5 because 0.5 is the largest mode. A network trained with the sum-of-squares cost function will also output a similar value being the average of the possible targets, which will also be close to 0.5. If a set of values of 0.5 started a time series then a network performing iterated prediction would make predictions of only 0.5 for each successive time-step. The network would give a correct output only a third of the time. It is only when the number of time-steps used in the input is 4 or greater that each of the training patterns are unique. This leaves the last pattern containing a multi-modal transition as shown in Table 6.1.

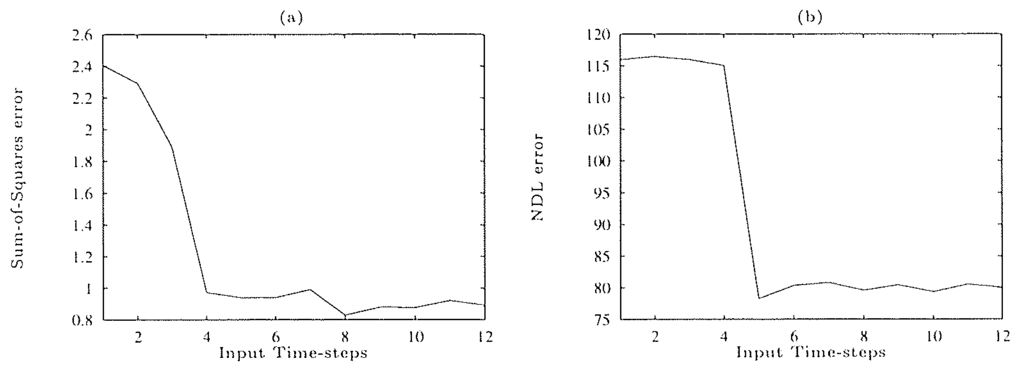

Figure 6.9: The training data error for the tri-valued time series data set. Networks were trained with different numbers of time-steps in the inputs. (a) A network trained with the sum-of-squares cost function. (b) A network trained with the NDL cost function.

Figure 6.9 shows the results of training with different numbers of time-steps in the input from 1 to 12 time-steps. The training set used was presented at the beginning of this chapter and contained no noise. Figure 6.10 shows how the test set NDL data error varies with the number of time-steps that are used in the input. The test set NDL data error was calculated for the networks trained using the sum-of-squares and the NDL cost function. The test set NDL data error for NDL training does not change until the number of time-steps in the input exceeds 4. This is because the number of time-steps covers the change from one value to another. For the tri-valued series the error decreases when the number of time-steps can capture when the transitions occur. A network may again learn only one transition leading to iterated prediction where only one output would ever be used. At best the network will make a correct prediction with a probability $P(\text{Correct Prediction}) = \frac{1}{3}$ as described already. If the network inputs contain information on when the multi-modal transitions occur then the probability is $P(\text{Correct Prediction}) = \frac{1}{2}$.

In this experiment the transitions are regular and inspection of the data can easily indicate the number of time-steps in the inputs to use. This was determined by looking at the data and noting when the transitions took place. Training was performed with no regularisation with the aim of fitting as many of the points as possible. For problems where there is noise in the data, generating graphs such as Figure 6.10 is a greater

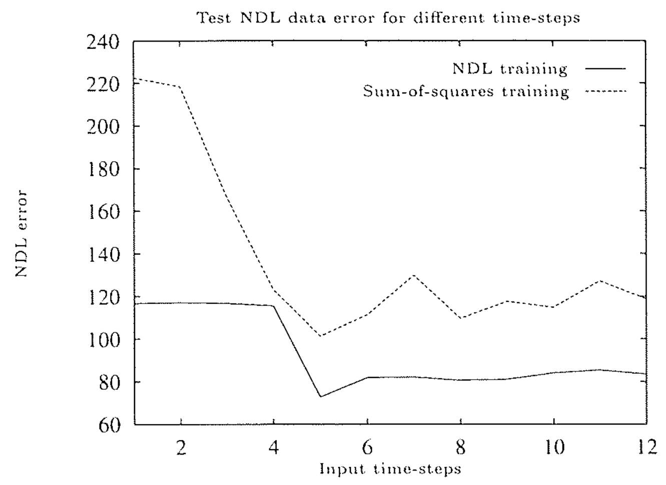

Figure 6.10: The NDL data error for the tri-valued time series test set. Networks were trained with different cost functions for a range of time-steps.

problem. This is because the minimum test data error must be obtained for each number of time-steps used in the input. Generating an α-survey is computationally expensive itself without having to generate many of these surveys to discover if different time-steps yield better results. Training with different numbers of time-steps in the network inputs meant that each network had to be trained from scratch, unlike the α-survey experiments. Training with completely different sets of weights in each case may lead to local minima problems. For the results presented here, for the problem containing no noise, 3 different time-step surveys were performed. A time-step survey consisted of training the network with a range of time-steps used on the inputs. The average of these 3 time-step surveys was used to obtain the graphs shown in Figures 6.9 and 6.10.

6.3.1 Time-steps and the amount of data

The 'inverse' XOR problem was used to further investigate the effect of the number of time-steps used in the training inputs. For this problem a single time-step would be enough with NDL training to get the desired results. As the number of time-steps in the input increases, the probability that a particular input pattern occurs just once in the time series increases. The sum-of-squares cost function will only form an average when two or more input patterns are the same and the targets are different. As the number of patterns in the data set gets higher the probability of a unique input is lower.

Figure 6.11 shows how the number of time-steps used affected the result. It can be seen from the graph that both methods converge for a small data set. The NDL

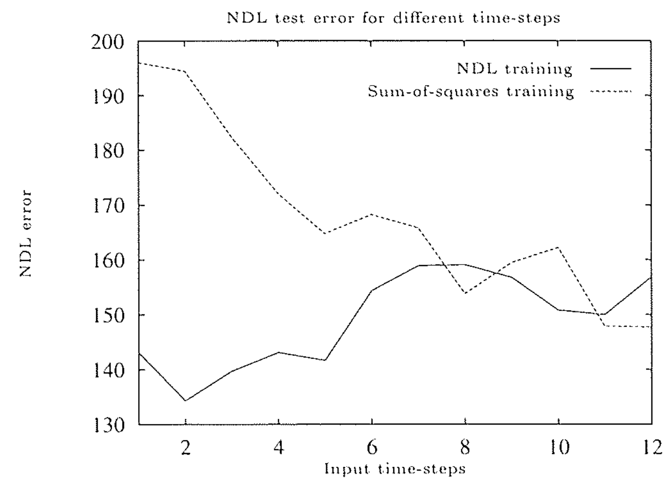

Figure 6.11: The NDL data error for the 'inverse' XOR test set. Networks were trained using the NDL and sum-of-squares cost function for a range of time-steps in the inputs.

cost function can achieve a better network mapping to a problem with less time-steps. Training with the sum-of-squares cost function will learn the mean giving a large NDL test error. The underlying process is a multi-valued problem and so the NDL cost function is better able to pick out this feature. With only 1 time-step, NDL training has been able to find a possible network mapping. The sum-of-squares cost function can only give the correct result when the number of time-steps is large enough so that two similar patterns with different target patterns do not occur. As the number of time-steps increases the generalisation required to unknown patterns increases and the performance of the NDL trained network begins to decrease.

6.3.2 Using the results from iterated prediction

Section 6.2 showed that plotting the error at two time-steps gave some indication that the error at one time-step was closely related to the error at the next time-step. It was shown that a known error indicated the size of an unknown error of iterated prediction in the future. If the error was large then the network was obviously following the wrong path and the output of the network should be ignored. A number of networks trained on each mode of the problem may be able to solve the problem of being on a path that is clearly the wrong one. Another network trained on another mode will predict more accurately for that mode. This may lead to an ensemble of networks.

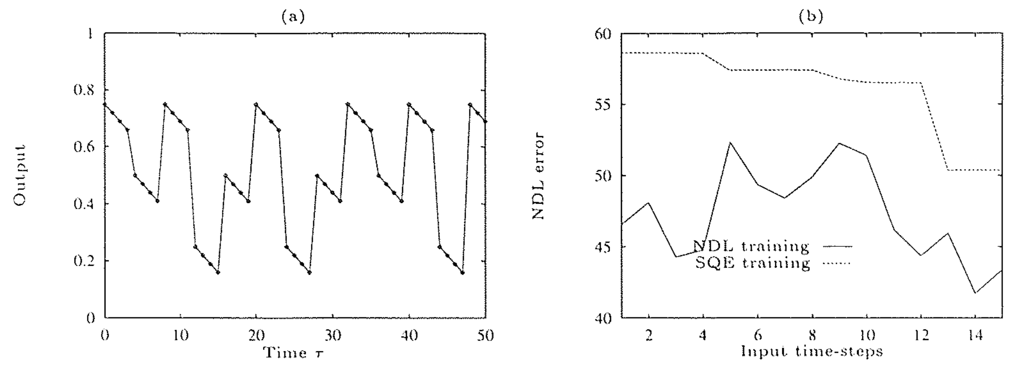

Figure 6.12: The modified tri-valued time series training set and results. (a) The training set for the modified tri-valued problem. (b) The NDL test error for NDL training and sum-of-squares cost function training.

6.3.3 Reducing the number of time-steps

In the tri-valued data set the error decreases when the number of time-steps can capture when the transitions occur. In this experiment an indication of when a transition will occur was included as part of the data. Instead of each group of 4 values being the same, they were different with each output decreasing slightly from the last. This data set shows that different problems have different kinds of repeating patterns. In this case the NDL cost function can give good results for a single time-step because all the inputs are unique. Figure 6.12 shows the data set and the results of training with different numbers of time-steps. As more time-steps are used in the network inputs the test set data error decreases for sum-of-squares training. The error can be seen to drop after another group of 4 inputs are used. The NDL trained network gives good results for a single time-step showing that for such problems the NDL cost function can give good test results with fewer time-steps.

6.4 NDL applied to some real-life problems

The last section showed the basic features of a time series containing multi-modal paths or transitions. This section presents results from using the NDL cost function on some real life data sets. The aim of using these data sets was to see if the NDL cost function could find any multi-valued paths or transitions in the data. The NDL cost function was applied to data of Bach's last Fugue, the Sun-spot data set and the Lorenz Attractor.

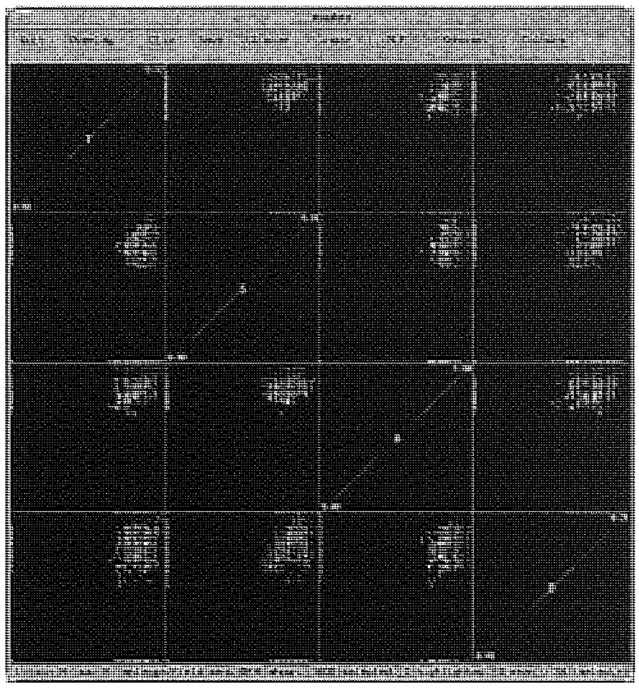

Figure 6.13: A Maden overview of the 3808 vectors in the Bach's last Fugue data set.

6.4.1 Bach's last Fugue

The NDL cost function was tested on a data set provided by the Santa Fe Time Series Competition. The data set comes from J.S. Bach's last Fugue, the uncompleted Contrapunctus XIV from *Die Kunst der Fuge* and was included in the competition because of the high amount of structure it contained. When we listen to music we have many expectations of the form and structure of the music. The data set provided focuses on the notes themselves and contains 3808 vectors. Thses vectors are 4 dimensional and give the pitch of the four voices in the music as a function of time. The units of time in the set are sixteenth notes or semiquavers. The four voices correspond to S (Soprano), A (Alto), T (Tenor) and B (Bass). A zero value in the data set represents a rest. The aim of including this particular data set in the competition was to see if computer assisted analysis and the use of Neural Networks could extract structure or themes in the piece of music. A Fugue contains a primary theme and may contain one or more secondary themes. Neural Networks may also discover any thematic transformations and when they occur. For more details see Dirst [13].

Figure 6.13 shows a Maden overview of the Bach's last Fugue data set. The Maden overview shows 16 separate graphs where in each column a particular dimension is plotted against each of the other dimensions. The diagonal shows each dimension plotted against itself so it is left blank. For example the first column of the second row shows S plotted against T. Figure 6.13 shows three clear clusters in the data. One large cluster in the top right of each plot and two narrow bands in the bottom right and top left. The purpose of using the NDL cost function in this example would be to more accurately predict a transition from one cluster to the next. This is preferable to predicting an average transition to a number of other clusters. No attempt was made to complete the Fugue.

To reduce training times the first 400 vectors in the data set were used as a training set and the following 400 vectors used to make up a test set. This reduced training set allowed α-surveys to be generated relatively quickly to show any multi-valued features in the data. The Maden overview of these sets showed the same number of clusters as shown in Figure 6.13.

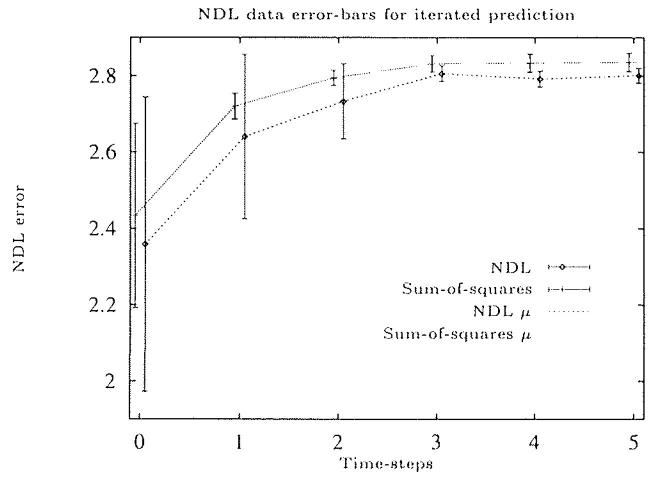

Figure 6.14: The mean NDL data error and its error-bars for two differently trained networks with no regularisation for the Bach's Fugue training set. (a) A network trained with the sum-of-squares cost function. (b) A network trained with the NDL cost function.

Figure 6.14 shows the mean NDL error and its error-bars for the iterated prediction errors of two networks trained with the sum-of-squares and NDL cost function. These networks show the results from training with no regularisation at all to discover any real multi-valued mappings. This is to see if two inputs that are exactly the same have two different outputs. This is different from multi-valued mappings, which consist of clusters of targets that exist for a range of inputs, for example the two-lines problem in Section 3.4.5. This was found useful to discover if multi-valued mappings existed in a data set before generating whole α-surveys for the set. In this case the network was allowed to overfit the data completely. The results show that there are a number of multi-valued mappings in the data set. This is because the mean NDL data error is smaller than the sum-of-squares data error for the first few iterated predictions in

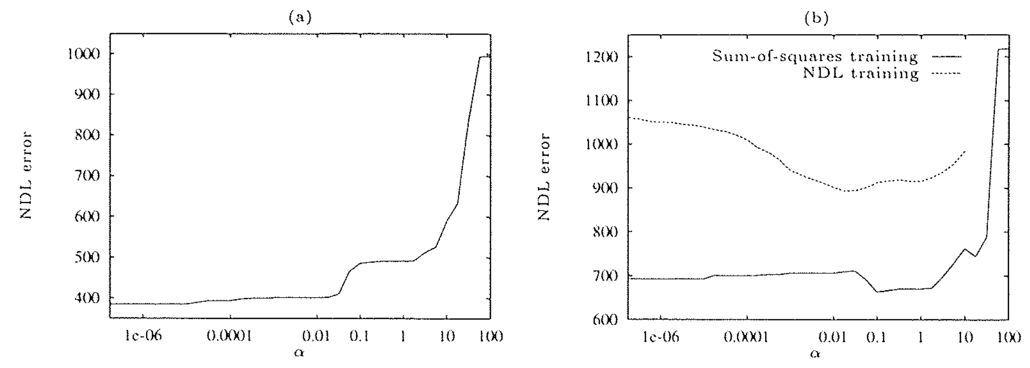

Figure 6.15: Training and testing α-surveys for the Bach's Fugue data set. (a) The training NDL data error showing two levels of structure. (b) The NDL test data error for the networks trained with the sum-of-squares and NDL cost function.

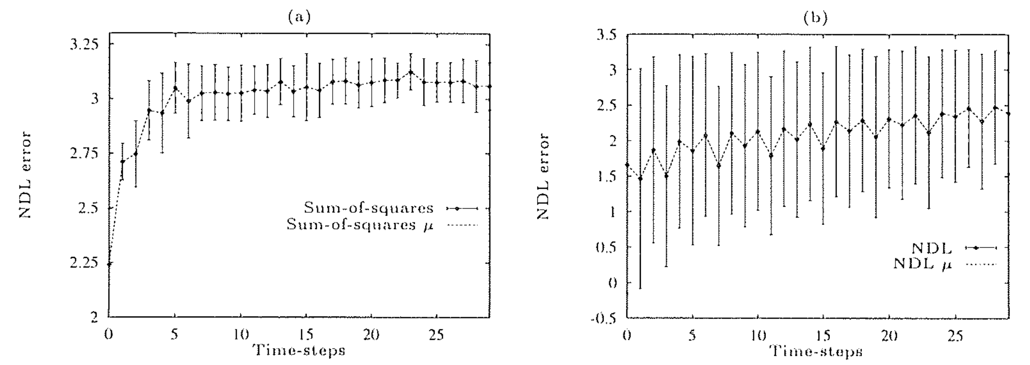

Figure 6.16: The mean NDL data error and its error-bars for the Bach's Fugue test set iterated prediction sequences. (a) A network trained with the sum-of-squares cost function. (b) A network trained with NDL cost function.

Figure 6.14.

The α-surveys for this problem were again generated using a standard MLP with 16 hidden nodes. Coarse-coding was used on normalised inputs between 0 and 1. A single time-step was used in training. Figure 6.15 shows the training and testing α-surveys for the Bach's Fugue data. Training using the NDL cost function showed two distinct 'cliffs' in Figure 6.15a, indicating different levels of structure. The test error where one of these 'cliffs' occured also approximately points to the smallest NDL test error. A network was also trained and tested with the sum-of-squares cost function. Large values of α in the α-survey caused problems in network training so the network was trained with a slightly smaller range of α values than the network trained with NDL. Figure 6.15b shows that the minimum error was much less for NDL training.

The networks that gave the minimum test set data error were used to perform

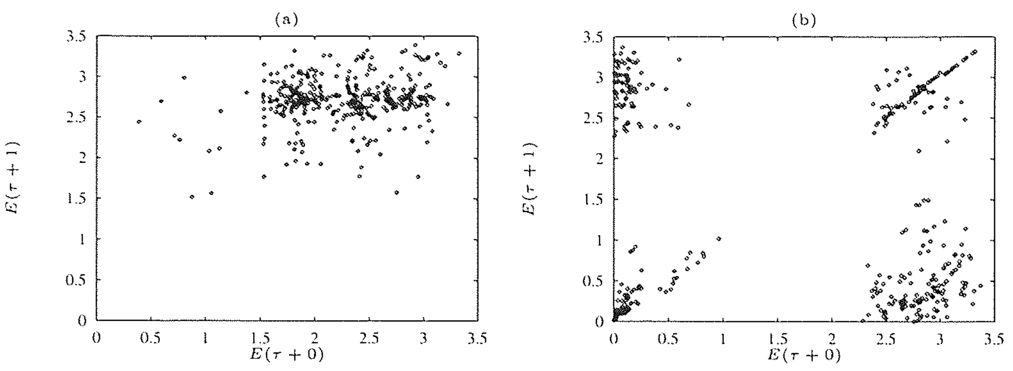

Figure 6.17: The NDL data error at different time-steps in an iterated prediction sequence for the Bach's Fugue test set. (a) A network trained using the sum-of-squares cost function. (b) A network trained with the NDL cost function.

iterated prediction. The results of iterated prediction are shown in Figure 6.16. The error increased rapidly for the network trained with the sum-of-squares cost function and reached a mean of approximately 3.0. The NDL trained network showed a smaller mean error at around 2.5 and the error increased more slowly. This meant that the output of the network was following more closely the value of actual values in the test data. When the network follows a particular path the error will be greater in places where the iterated prediction has followed the wrong path in the data. This is shown by the wider error-bars in Figure 6.16b.

Figure 6.17 shows the first and second time-step errors for iterated prediction. In the NDL trained network there are 4 clusters corresponding to 4 types of error. The cluster of points in the bottom left of Figure 6.17b show predictions that have stayed on a particular path. Notice that these errors are smaller than the test set NDL errors produced from a network trained with the sum-of-squares cost function in Figure 6.17a. The two clusters in the top left and bottom right correspond to a switch in the path of the data. The network has predicted one path when the actual data followed another. The top right cluster shows the situation when the network has made a prediction following a switch to a path that the actual data has not made. By using more time-steps the network may be able to predict more accurately when the transitions occur and hence reduce the size of the clusters in the top left and bottom right. By using more time-steps in training, the clusters in Figure 6.17 may become densest in the two clusters along the $x = y$ line.

Figure 6.18 shows the error-bars for two networks trained using 2 time-steps instead of 1. Again the α-survey was generated and the minimum test error used to choose a network to perform iterated prediction. Again the error-bars look similar to those produced for 1 time-step. Training with 2 time-steps shows a slightly different pattern

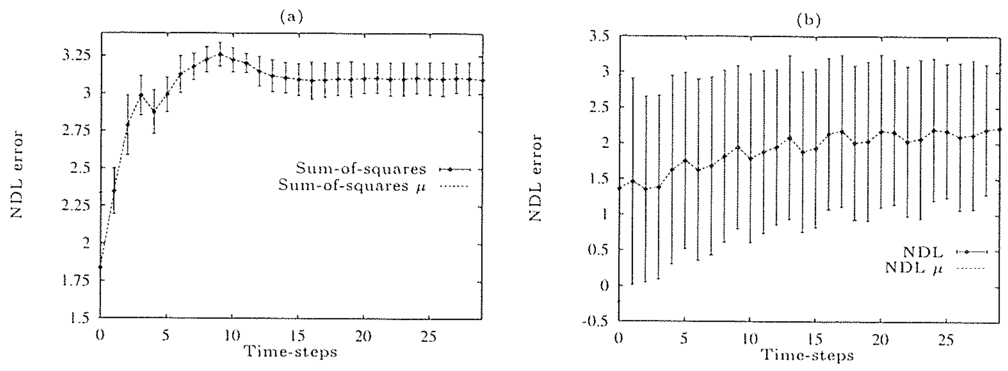

Figure 6.18: The mean NDL data error and its error-bars for the Bach's Fugue test set iterated prediction sequences. (a) A network trained with the sum-of-squares cost function. (b) A network trained with the NDL cost function.

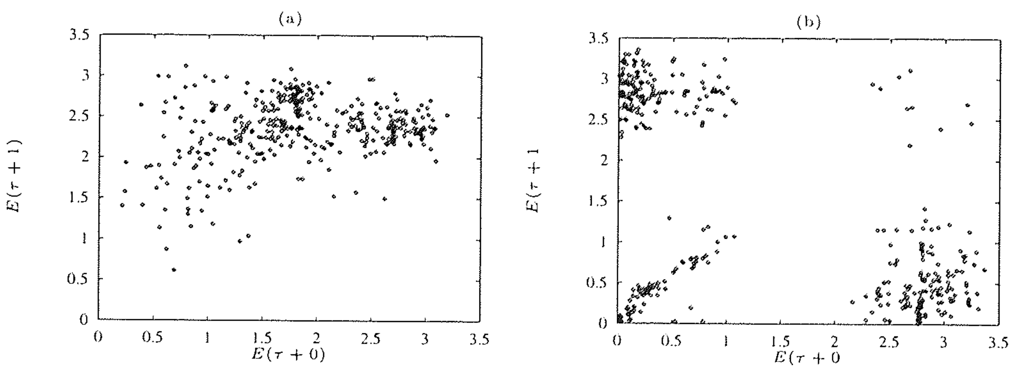

Figure 6.19: The NDL data error at different time-steps in the set of iterated prediction sequences for the Bach's Fugue test set. (a) A network trained using the sum-of-squares cost function. (b) A network trained with the NDL cost function.

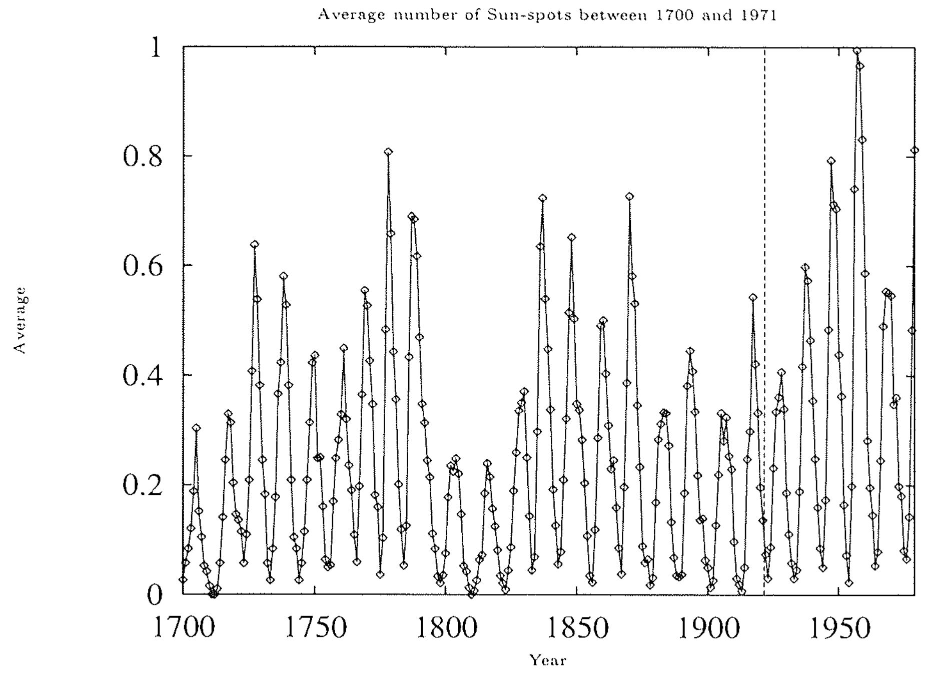

Figure 6.20: The average number of Sun-spots between the years 1700 and 1971. The training data set contained data from the years 1700 to 1920. The test set contained data from the remaining years 1921 to 1979.

of prediction errors between consecutive time-steps in the iterated prediction sequence. Figure 6.19 shows the iterated prediction errors at two time-steps. The pattern of errors for the network trained with sum-of-squares looks the same. For NDL training the top right cluster of points is smaller. This indicates that once the network has made a wrong prediction the next prediction is unlikely to be bad. More points appear in the bottom left cluster indicating good predictions in two consecutive time-steps. However more points have appeared in the top left and bottom right indicating that the next prediction is more likely to be wrong after a correct prediction. This is opposite to what was desired. In this case an exhaustive survey of time-steps would be needed to find clusters that lie only on the $x = y$ line. This survey is not presented here.

6.4.2 Predicting Sun-spots

The NDL cost function method of training was tested on the problem of predicting Sun-spots. NDL training may be useful in this problem if the pattern of Sun-spots follows a number of regular patterns that occur at random. NDL cost function training will pick this feature out of the data if it is there. For a more detailed description of the prediction of Sun-spots see Weigend [88]. The data contains recorded observations of Sun-spots from the year 1700 to 1979 and is a popular benchmark for time series

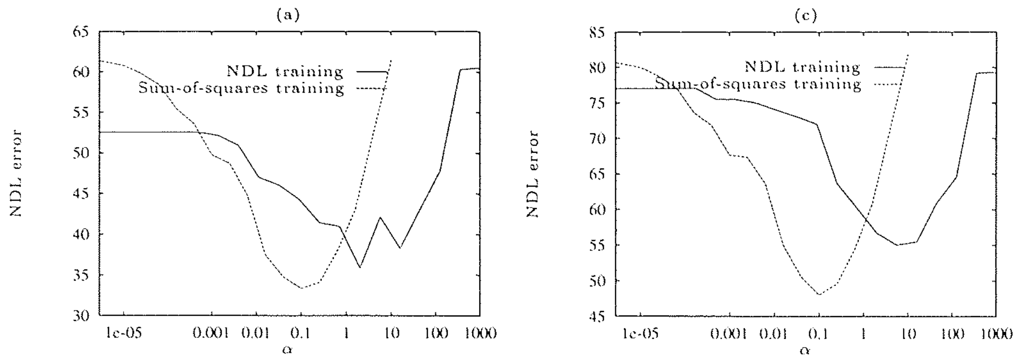

Figure 6.21: Testing networks trained with the NDL and the sum-of-squares cost function for the Sun-spot data. Networks were trained with different values of ϵ. (a) A network trained and tested with $\epsilon = 0.05$. (b) A network trained and tested with $\epsilon = 0.03$.

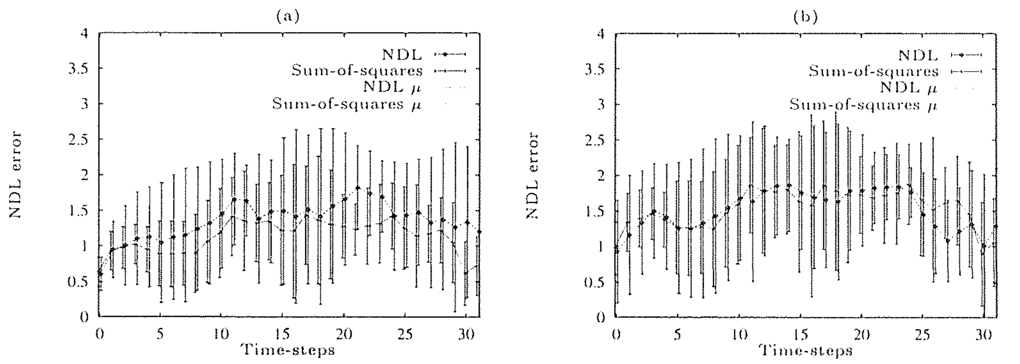

Figure 6.22: The mean NDL data error and its error-bars for the Sun-spot test set iterated prediction sequences. (a) A network trained and tested with $\epsilon = 0.05$. (b) A network trained and tested with $\epsilon = 0.03$.

prediction systems. Sun-spots were recorded by both Tong and Preistly. The training set contains observations from 1700 to 1921 and the test set contains observations from 1921 to 1979.

As described in Section 6.3 it is quite difficult to determine the correct number of time-steps to use to train a network. In this case the number of time-steps was chosen from a simple inspection of the data. It was noticed that the number of Sun-spots increased and then decreased in a fairly constant cycle of about 12 to 14 years. Therefore 12 years were used as the number of time-steps for training. This would capture whether the size of the peak of the number of spots determines what happens in the next cycle. It may be that there is some deterministic pattern occurring for a larger range of years but conducting such surveys of the data would seriously affect the training times.

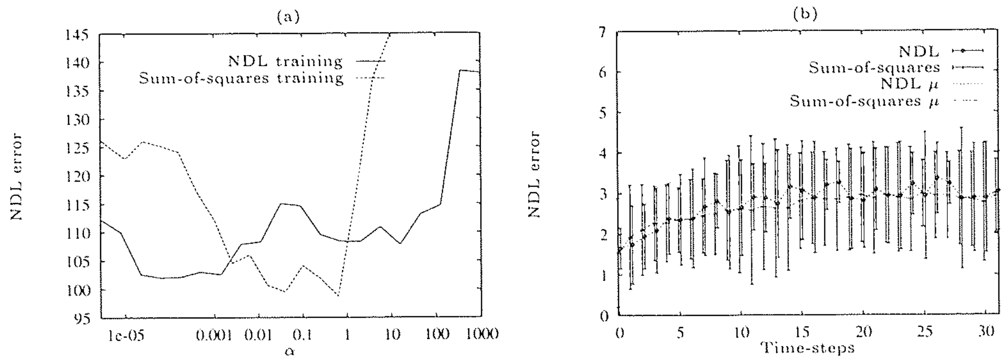

Figure 6.23: The results of training with 6 time-steps instead of 12 on the Sun-spot training set. (a) The test set data errors for networks trained with the NDL and sum-of-squares cost function, where $\epsilon = 0.01$. (b) The mean NDL data error and its error-bars for the two differently trained networks.

All data was normalised and trained with a conventional MLP using coarse-coding on the input time series. For this problem two networks were trained, one using the conventional sum-of-squares cost function and the other using the NDL cost function. Figure 6.21 shows the results from a test set where the network was trained with the two training methods and tested with the NDL cost function. Figure 6.21a shows that training using the sum-of-squares cost function achieved a smaller NDL data error than training with the NDL cost function. Training with the two methods showed no significant difference in performance. Training with a slightly smaller value of ϵ was carried out but this again made little difference. The NDL test error for the network trained using sum-of-squares was still smaller. The mean NDL data error and error-bars for networks that gave the minimum test set data errors showed no difference in iterated prediction performance shown in Figure 6.22.

It is thought that the disappointing performance may have been due to too many time-steps being used in the example. The training set does not have many values so training with a large number of steps will degrade the generalisation capability of the network. This was seen in the 'inverse' XOR problem and the tri-valued regular time series problem in Section 6.3.1 and 6.2. The experiments were therefore repeated with 6 time-steps.

Figure 6.23 shows the results of training with 6 time-steps instead of 12. Again the two methods of training are similar and indicate that there are no multi-modal parts to the data. However such parts cannot be ruled out without training over a large range of time-steps.

6.4.3 The Lorenz Attractor

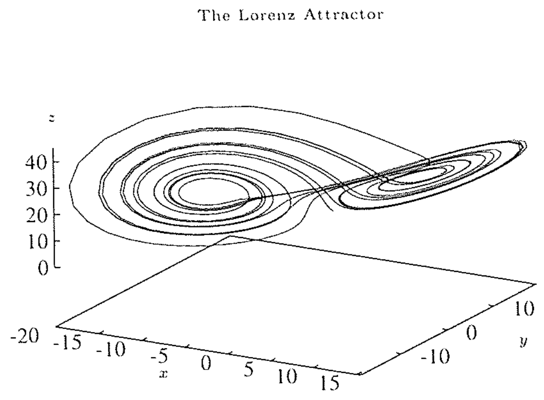

Figure 6.24: The two Basins of Attraction of an orbit generated by using the Lorenz Attractor equations.

In 1963, Edward Lorenz published a paper on deterministic chaos describing a set of equations giving orbits in 3-dimensional space containing an attractor with two lobes [45]. From an initial point an orbit can be generated such as the one in Figure 6.24. Orbits like these are sometimes used to generate Fractals. The orbit spirals around one of the lobes and at particular points breaks away and spirals around the other lobe. These lobes are sometimes called Basins of Attraction. Generating points in the orbits shows that part of the time the orbit spirals around one lobe and part of the time spirals around the other. The movement of the orbit around one of the lobes can be seen as the deterministic part of the data. The sudden movement to a different lobe as the chaotic or random part of the data. For an excellent introduction to Basins of Attraction and Fractals see Wegner *et al.* [87].

Given a starting point in 3-d space the equations shown below were used to generate a training and test set. The Lorenz Attractor equations are:

$$x' = x - (ax + ay)dt\,, \tag{6.1}$$

$$y' = y + (bx - y - zx)dt\,, \tag{6.2}$$

$$z' = z + (xy - cz)dt\,. \tag{6.3}$$

Each value of x, y and z were calculated from the preceding values. The values of a, b and c were set at 10.0, 25.0 and 2.67 respectively. The time interval dt was set at 0.02. The training set was initialised with x, y, and z with values of (-1.0, -1.0, -1.0) and 1100 points were generated. The first 100 points were discarded to allow the

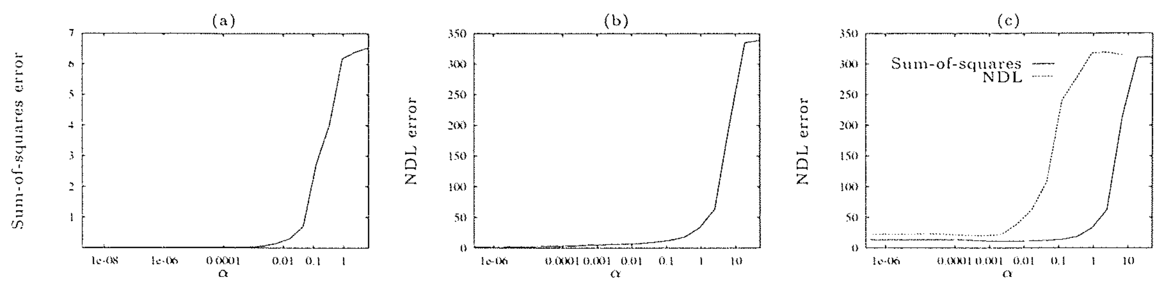

Figure 6.25: Training and testing α-surveys for the Lorenz data set. Networks were trained with the sum-of-squares and NDL cost function. (a) The training α-survey using the sum-of-squares cost function. (b) The training α-survey using the NDL cost function. (c) The test set NDL data error for the networks trained using the sum-of-squares and NDL cost function.

orbit to move onto one of the lobes. The next 500 points were used for training and the remaining 500 points used for testing. To reduce training times every other point was discarded giving training and test sets with 250 points. Figure 6.24 shows both training and testing patterns. The data is clearly multi-modal in that an orbit can either continue to spiral around one lobe or to spiral around the other lobe. The aim of using the NDL cost function was to see, given an initial starting point around one of the Basins of Attraction, if iterated prediction of an orbit would continue around one lobe only.

A 2-layer MLP was used with 15 hidden nodes. The number of time-steps included in the network inputs was 5. No experiments were carried out to optimise the number of time-steps for this problem. The sum-of-squares and NDL cost function were used to generate α-surveys, the value of ϵ was set at 0.05. Each network was trained over a range of values for α. It was important to make sure that each network trained so that the total error at small values of α approached zero. This would make sure that all the points were fit so that all the levels of structure can be seen in the α-survey. Figure 6.25a and 6.25b show the α-surveys for sum-of-squares and NDL cost function training. In Section 4.2 it was shown how the training set alone could be used to set the value of α. In this experiment it was found that RBF networks were not good enough for prediction and so a standard MLP was used. This meant that training times would take much longer so to avoid long training runs each network was not trained to completion. Results from using RBF networks are shown later in this section.

Figure 6.25c shows the NDL test set error for each of the networks. The network trained with the sum-of-squares cost function was trained over a slightly different range from the networks trained with NDL. This was because large values of α resulted in some of the weights sharing very small values. This meant that they did not train very

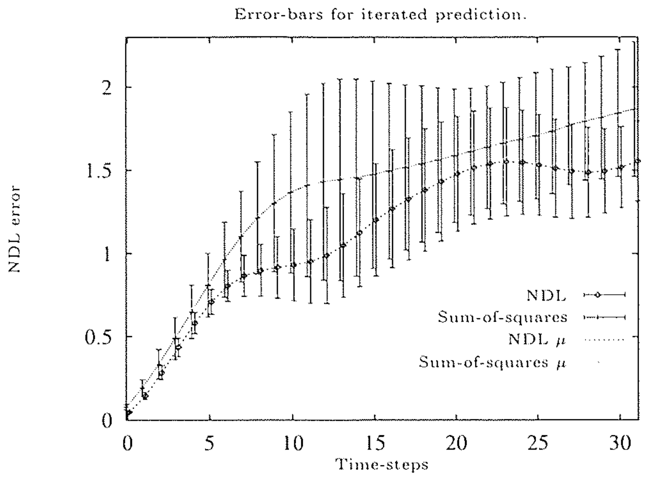

Figure 6.26: The mean NDL data error and its error-bars for the Lorenz test set iterated prediction sequences. Networks were trained with the NDL and sum-of-squares cost functions.

well for smaller values of α (see Section 4.1 for details on the α-survey and its potential problems). The network trained with the NDL cost function shows the underfitting and overfitting nature of the training with the minimum test set error being 11.007 with an α value of 0.006. This is compared to 19.791 at an α value of 0.008 achieved by the network trained with the sum-of-squares cost function. At each of these values of α, iterated predictions were made.

Figure 6.26 shows the prediction NDL data errors for networks trained with the sum-of-squares and NDL cost function. The mean error for the NDL trained network is smaller than that for the sum-of-squares cost function trained network. In a problem such as this the number of times that the orbit moves from one lobe to the other is small compared to the number of spirals around a single lobe. Therefore the difference in the NDL error for each trained network was not expected to be very large.

Figure 6.27 shows the error at two time-steps plotted against each other for the two networks for the set of iterated prediction sequences. The network trained with the NDL cost function does show slightly smaller errors overall than the network trained with the sum-of-squares cost function. It has been already mentioned that the number of times the orbit switches to a different lobe is small. The Lorenz Attractor equations also cause loops that never repeat. Starting the orbit with different initial values will always generate completely different paths. The training set only contained 250 patterns, the next 250 points in the orbit may look the same but follow a different path. This may explain why for very small values of α in training the NDL test set data error did not increase or overfit very much. Though small differences in NDL data

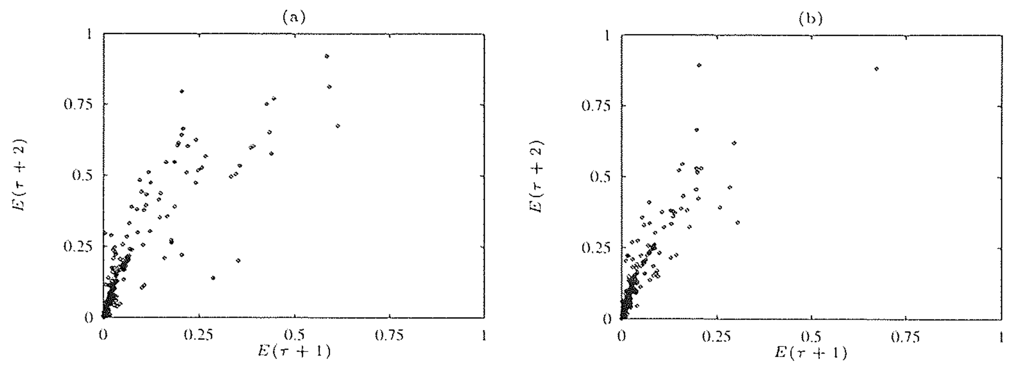

Figure 6.27: The NDL data error for different time-steps from time τ for the Lorenz test set iterated prediction sequences. (a) A network trained with the sum-of-squares cost function. (b) A network trained with the NDL cost function.

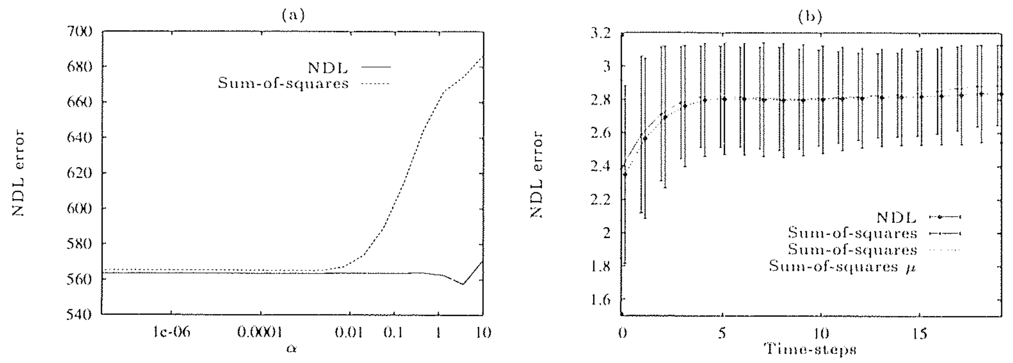

Figure 6.28: RBF network results for the Lorenz data set. (a) The test set NDL error for networks trained using the Adaptive NDL and sum-of-squares cost function. (b) The NDL data error mean and its error-bars for the set of iterated prediction sequences.

error were detected for this data set, larger differences might be found with networks trained with much bigger training sets. This obviously increases training times, which was a major restriction in training with larger data sets for the results presented here.

RBF networks were also used to do iterated prediction for this problem. However the results obtained with this faster method were not as good as with a 2-layer MLP. Networks were trained with the same range of α values in the α-survey and the same ϵ value of 0.05 for the NDL training. In each case the error approached zero being closer to zero than any of the errors from networks trained with a standard MLP. However the test set errors were larger. The test set mean error and its error-bars showed no difference between the two methods of training. A difference was only found when Adaptive NDL was used. Adaptive NDL is described in Chapter 7 and is a method of NDL training where the value of ϵ becomes much smaller than for all the experiments shown so far. It involves decreasing the value of ϵ during training. Figure 6.28a shows the results of training for Adaptive NDL and testing with ϵ set to 0.01. The minimum test set data errors were located and their α values used to do iterated prediction. Figure 6.28b does show a slight difference in the two methods for the test set NDL data error mean and its error-bars. The RBF network does not generalise very well for this problem where the attractor orbits do not repeat.

Using Adaptive NDL does suggest a possible improvement to training where data sets have more points. In the data set used for these experiments every other point was taken out of the data set generated using the Lorenz Attractor equations. This meant that movement of the orbit to other lobes would be clearly seen. As the value of dt in the Lorenz Attractor equations decreases the changes may be more subtle. In these cases very small values of ϵ may be needed to distinguish between the two general paths that the orbit can take.

6.4.4 Summary

NDL was applied to time series problems. A simple tri-valued time-series was presented. The NDL cost function was able to pick a mode in the multi-valued transitions present in the time series. The NDL cost function caused a network to converge to the most likely targets in the time series instead of the mean of all the possible targets. In iterated prediction this caused the network to stay on a particular 'path' more accurately. The error was greater however when the wrong path was chosen by the network.

The number of time-steps was found to have a large affect on the generalisation ability of any trained network. They were also able to identify points in the time series where the random transitions in the data could be identified. For problems where the random transitions in the data could be determined from a single time-step, additional time-steps were used. This was done to compare NDL data error with sum-of-squares data error for the two methods of training. As the number of time-steps increased, the performance of sum-of-squares cost function training improved until the two methods converged. The sum-of-squares data error decreased because the more time-steps there were in the input, the less likely it was that two input patterns would have multi-valued outputs for a fixed size data set. The NDL data error grew worse for more time-steps in the input because more unseen patterns appear in the test set.

NDL was applied to a number of real-life time series problems. Good results were achieved for iterated prediction on the Bach's Fugue data set and the Lorenz Attractor data set using NDL compared to sum-of-squares cost function training. Results showed that networks followed the most probable target or path from a number of possible targets or paths that could have been followed. Limitations in computing power meant exhaustive tests of the whole of the Bach's Fugue data set could not be done. This was also the case for a much larger Lorenz Attractor data set. However the results presented are encouraging and gives confidence that it would work for the larger data sets.

Chapter 7

Adaptive NDL

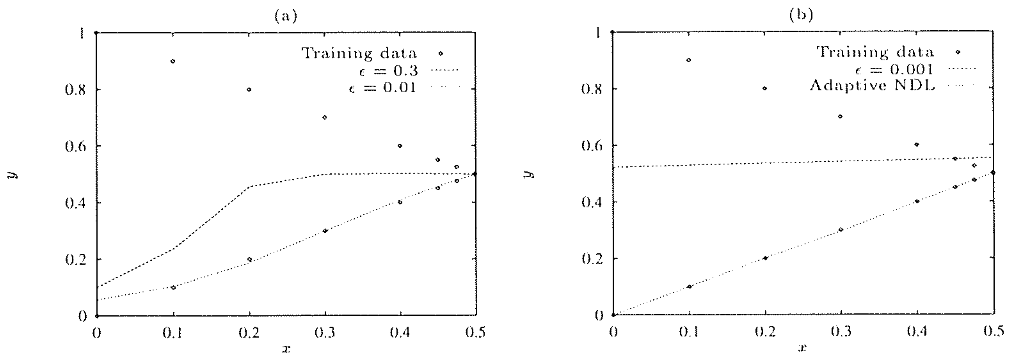

Figure 7.1: The problem with training networks with different values of ϵ. (a) Training with ϵ at a large value of 0.3 and an intermediate value of 0.01 which would be used to gain the best results. (b) Training with ϵ at a very small value of 0.001 and using a technique of changing the value of ϵ during training.

In most of the toy problems presented so far, where there is a multi-valued feature, the different values for each input have easily been distinguished by the value of ϵ used in training. For the multi-valued lines problem and the simple tri-valued regular time series problem the possible values that the output could have for a given input were very different. A value of ϵ was chosen that would capture one of the possible output values. It has already been shown that difficulties arise when a data set contains multi-modal data at different 'separations'. This section describes difficulties in the actual training of the network and what the network eventually converges to. A problem occurs in training when a data set contains a multi-valued feature where the different values are very close to each other. This could occur in data sets for which there is no prior knowledge of whether there is any multi-valued feature in the data. Setting ϵ too large would not detect multi-valued outputs at a small scale. One could argue that using a very small value for ϵ all the time would make sure that multi-valued outputs at a small scale would be captured. However, training is made more difficult as the problem presented here will show. Another motive for using small ϵ is that it can also represent large scale structure. This was explained in Section 4.6 where a small value of ϵ was able to produce network outputs that followed one of the clusters of data, even when the cluster varied in scale.

Figure 7.1a shows a training data set where there are multi-valued outputs at different separations. For an input of 0 the possible output is 0 or 1 with equal probability. The final input of 0.5 has a possible output of 0.501 or 0.499 with equal probability. An MLP was trained using the NDL cost function with different values for ϵ. When the value of ϵ was 0.3 the larger separation multi-valued outputs were captured but not the smaller separation ones. When the value of ϵ was 0.01 the result is better but the input pair of 0 to (0, 1) is not quite correct. When ϵ is set at 0.01 this is the best we will expect when using that value throughout the whole of the training. When the value of ϵ is really small at 0.001 as shown in Figure 7.1b, the network converged to fit only one of the points. This was for a value of ϵ that was expected to capture all the multi-valued outputs in the data set. This is because the random weights at the start of training give outputs which, when using the NDL cost function, are far away from the correct target value. The gradient in these regions is very flat causing training to be more difficult. Finally Figure 7.1b shows how a method of altering the ϵ scale during training can give the desired results.

Capturing multi-valued outputs at different scales is done by altering ϵ so that it starts at a large value and concentrates on large separation multi-valued mappings such as the input-output pair of 0 and (0, 1). Within the ϵ scale, training will cause multi-valued outputs to be trained roughly according to a sum-of-squares cost function. Training continues until the difference in total NDL data error between two epochs is within some tolerance limit, for example a value of 0.0005. This indicates that training has reached a minimum at this level of scale. The value of ϵ is then multiplied by a constant that decreases its value and training continues. In this way the outputs of smaller scale multi-valued mappings are forced to converge toward a mode instead of the mean of the possible outputs. The outputs for larger scale multi-valued outputs are pushed even more to one mode of the possible outputs. In training a network using this Adaptive NDL technique a scale factor of 0.9 was found to be a good one for the problem above. The starting value of ϵ was set to 0.1 in this problem but it has been shown in Figure 7.1 that 0.01 would be a good starting value. In practice the value of ϵ was not allowed to reach a value smaller than a certain limit, usually set at 0.0001.

An artificial data set was created to illustrate the same problem in a time series prediction task. The task is similar to the tri-valued regular time series problem described in Section 6.2. Here the possible transitions from one value to the next were

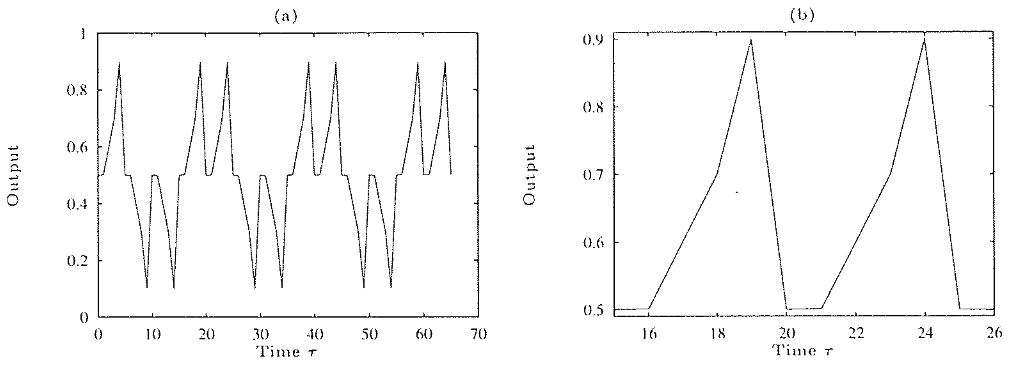

Figure 7.2: The training set for a two-path time series. (a) The whole training data set. (b) Part of the training set showing the small change that leads to one of the two different paths taken by the data.

Test	Total Error	Well-fit points
NDL, $\epsilon = 1$	106.014	51.490
NDL, $\epsilon = 0.0001$	453.732	13.735
Sum-of-squares	27.691	59.994
Adaptive NDL	17.979	61.048

Table 7.1: The NDL test data error for a variety of differently trained networks.

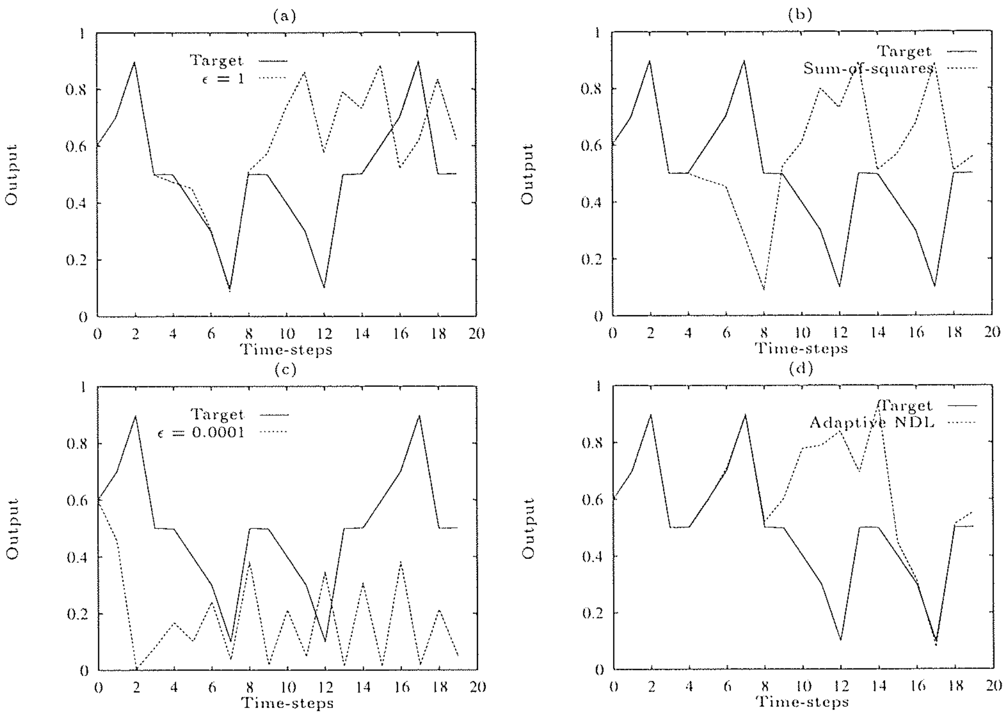

Figure 7.3: Iterated prediction sequences for networks trained with the NDL, Adaptive NDL and sum-of-squares cost function on the two path time series test set. (a) A network trained with the NDL cost function with $\epsilon = 1$. (b) A network trained with the sum-of-squares cost function. (c) A network trained with the NDL cost function with $\epsilon = 0.0001$. (d) A network trained with the Adaptive NDL cost function method.

clear and at a 'separation' where a value for ϵ could be easily chosen. In the time series problem that follows the idea of a repeating pattern is used again. This time the transitions that could lead to either path start with a very small change. When the individual time series outputs are examined a result similar to the above is found.

Figure 7.2a shows a training set where a value of 0.5 in the time series leads to either 0.501 or 0.499. Figure 7.2b shows part of the training data containing the two possible paths that can be taken. The resulting path taken depends on a correct mapping of these small separation multi-valued outputs. Table 7.1 shows the NDL data test error for training with different values of ϵ along with the NDL data test error for sum-of-squares and Adaptive NDL cost function training. Though these networks were trained with different values of ϵ, they were all tested with a single test ϵ so that the different results could be compared. When the value is allowed to change in Adaptive NDL cost function training the test error achieves a much smaller value than when ϵ is not allowed to change.

Figure 7.3a and 7.3c show iterated prediction sequences for training with a fixed value for ϵ of 1 and 0.0001. For a large value of ϵ of 1 the cost function is similar

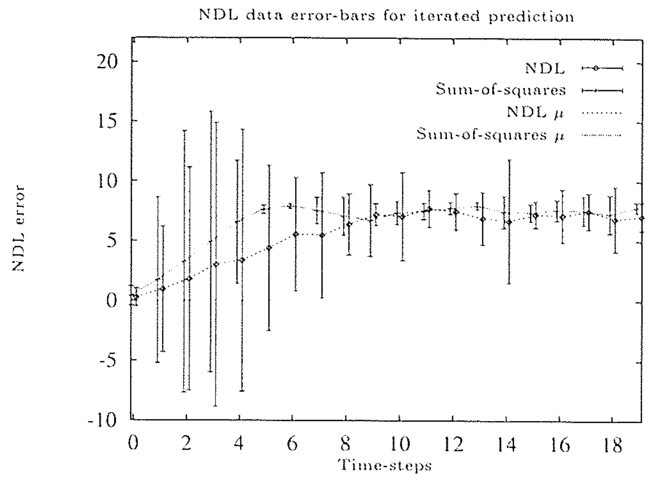

Figure 7.4: The two path time series test set mean NDL data error and its error-bars for the set of iterated prediction sequences.

to the sum-of-squares cost function. When the value of ϵ is very small at 0.0001 the network does not train properly. For the first 10 time-steps the prediction sequence closely follows one of the possible paths in Figure 7.3d, where the Adaptive NDL cost function is used.

The mean NDL test data error and its error-bars are shown in Figure 7.4. The mean NDL data error is smaller for the NDL trained network than the mean NDL data error for a network trained with the sum-of-squares cost function. After about 10 time-steps the mean error does tend to be the same for the two networks.

In the time series problem just presented a very small random change gave two deterministic paths that the training data could take. For this problem there was no noise present in the training data. For data sets with noise that have the same underlying feature it would be more difficult to get a network to follow one particular path. The changes that lead to different paths would be much smaller than the noise present in the data.

7.1 Summary

Data sets were presented where the difference in separation between the multi-valued targets for a given input ranged from very small to very large. NDL cost function training with too small a value of ϵ led to network outputs that did not fit any of the points. Training with too large a value of ϵ led to network outputs that were the mean of the multi-valued targets where these targets had small separations. This and the problem of small changes leading to a number of alternative paths in a time series prompted the development of Adaptive NDL. This was shown to perform better on problems with a variability in separation between the two modes of a given input. Adaptive NDL achieved better results on two simple toy time series problems.

Chapter 8

NDL in the weights

8.1 NDL weights as a regulariser

The previous chapters have used the NDL cost function to measure the error of the outputs from the network. This section takes the NDL principles and applies them to the weights in the network, giving them an information length interpretation. The different methods of using regularisation have been discussed in the review. Regularisation can be interpreted as controlling the amount of information available in the network to code a particular mapping from the inputs to the outputs. The conventional weight-decay term used in previous chapters restricts the possible weight space with which the network must model the data. This chapter views regularisation as restricting the information available in the weights of the network. In Equation 3.12 the sum of the error was calculated and the ϵ scale gave a measure of the scale or accuracy that the error could have. The cost function treated all errors outside the ϵ scale as similar. When using the NDL principle for the weights we would like many of the weights to be zero, so each weight of the network replaces the error term of Equation 3.12 in NDL regularisation.

$$E_{Total} = E_{Data} + \alpha E_{Weights} \tag{8.1}$$

$$E_{Weights} = \sum_i \ln\left(\frac{w_i^2+\epsilon^2}{\epsilon^2} \right) \tag{8.2}$$

Equation 8.2 shows the regularisation or network error. Again the parameter α determines how much influence the values of the weights make to the network solution. The gradient of the function expressed in Equation 8.2 is calculated in Appendix A.2. It has been shown that the number of 'well-fit' points can be calculated from using the NDL cost function to form models of the data. In the same way a measure of the number of 'well-eliminated' weights can be calculated, which gives the number of weights that approximate to zero,

$$N = \sum \frac{\ln\left(\sqrt{w_i^2 + \epsilon^2}\right)}{\ln(\epsilon)} . \tag{8.3}$$

Weigend *et al.* [89] have also developed a term that can be added to the data cost function called weight elimination. This controls the network complexity using the ideas of minimum description length but has no direct interpretation as a description length. It has the following form and is very similar to Equation 8.2:

$$E_{Total} = E_{Data} + \alpha \sum_{i \in C} \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2} . \tag{8.4}$$

The network term measures the size of the net over all the weight connections. Again α is a regularising parameter. In this equation w_0 is a parameter like ϵ and must be chosen by hand. For weights much larger than w_0 the cost for each weight will approach 1. For weights much less the cost will approach 0 so the sum can be interpreted as the number of large weights relative to w_0. However, the NDL weights regulariser in Equation 8.3 gives large negative errors when weights approach zero. Therefore, training with the NDL weights regulariser will work harder at eliminating small weights.

Some weights in a network may be redundant, for example if a single network node is connected to two equal weights from an input node. Minimising the information or description length of the weights would cause one weight to be driven to zero or eliminated. The parameter w_0 allows the possibility of preferring fewer larger weights with a small value of w_0 or many small weights with a larger value of w_0. The ϵ term in Equation 8.2 will be shown to have a similar effect. The new network cost function presented here views the weights by their 'closeness' to zero. It also introduces the logarithm to make sure that weights that cannot be eliminated at certain values of α have no effect. This section shows what effect the new network complexity cost function has when α surveys are generated.

For the experiments described here the same α-surveys were generated. The parameter α was allowed to decrease by a number of steps with the final weights of one training run used to initialise the training for the next value of α. In NDL data training it was seen that the decrease in cost of fitting a point within the ϵ scale was greater than the increase in cost incurred for moving away from a point outside the ϵ scale. This was shown by the simple problem presented in Section 3.3.4. In this problem there was a clear multi-modal feature in the data. However this feature is not present in the weights. The previous explanation for NDL data training can be re-phrased for NDL in the weights. The decrease in cost of fitting a weight within the ϵ scale

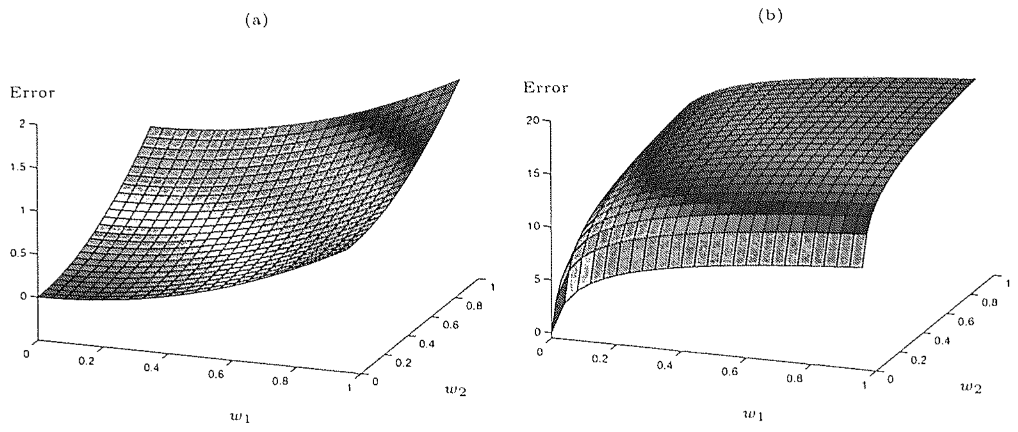

Figure 8.1: How the network error changes for varying weights. (a) For weight-decay. (b) For NDL weights with weight $\epsilon = 0.01$.

is greater than the increase in cost of leaving a weight outside the ϵ scale. This will have a greater or lesser effect depending on the value of α. For large values of α in the α-survey the aim is to drive all the weights to values within the ϵ scale, so it is quite similar to weight-decay within that scale. At smaller values of α some weights may become 'free' to model the data while other weights still fit within the ϵ scale. Figure 3.5 showed the individual error of two outputs and their sums according to the NDL data cost function. It showed two minima corresponding to each of the two possible output values. When using NDL in the weights these two errors can be replaced by two separate weights representing a very simple network architecture.

Figure 8.1 shows how the network error varies as the values of each of the weights vary. Figure 8.1a shows that as the weights get larger the gradient of the surface gets steeper so training will cause large weights to be penalised. For the NDL weights regulariser the gradient flattens out for large weights in Figure 8.1b.

To demonstrate the properties of NDL regularisation a simple problem was chosen where the training set contained two input patterns, (0,0,1,0,0) and (0,0,0,0,0). These gave two target patterns of 1 and 0. In this problem an ideal network can easily be drawn by hand. It would be a single layer perceptron model of 5 inputs feeding into 1 output node. Simple inspection of the data tells us that the target depends on the 3rd component of the input and nothing else. An ideal network would have weights that are 0 for all inputs that are 0 in the pattern (0,0,1,0,0). The 3rd weight would be large enough to cause a sigmoid function to saturate to give an output of 1. The output bias would similarly be negative. Training a network over a range of α using the sum-of-squares cost function and weight-decay regularisation will give non-zero weights at small values of α. By increasing weight-decay regularisation all the weights will be

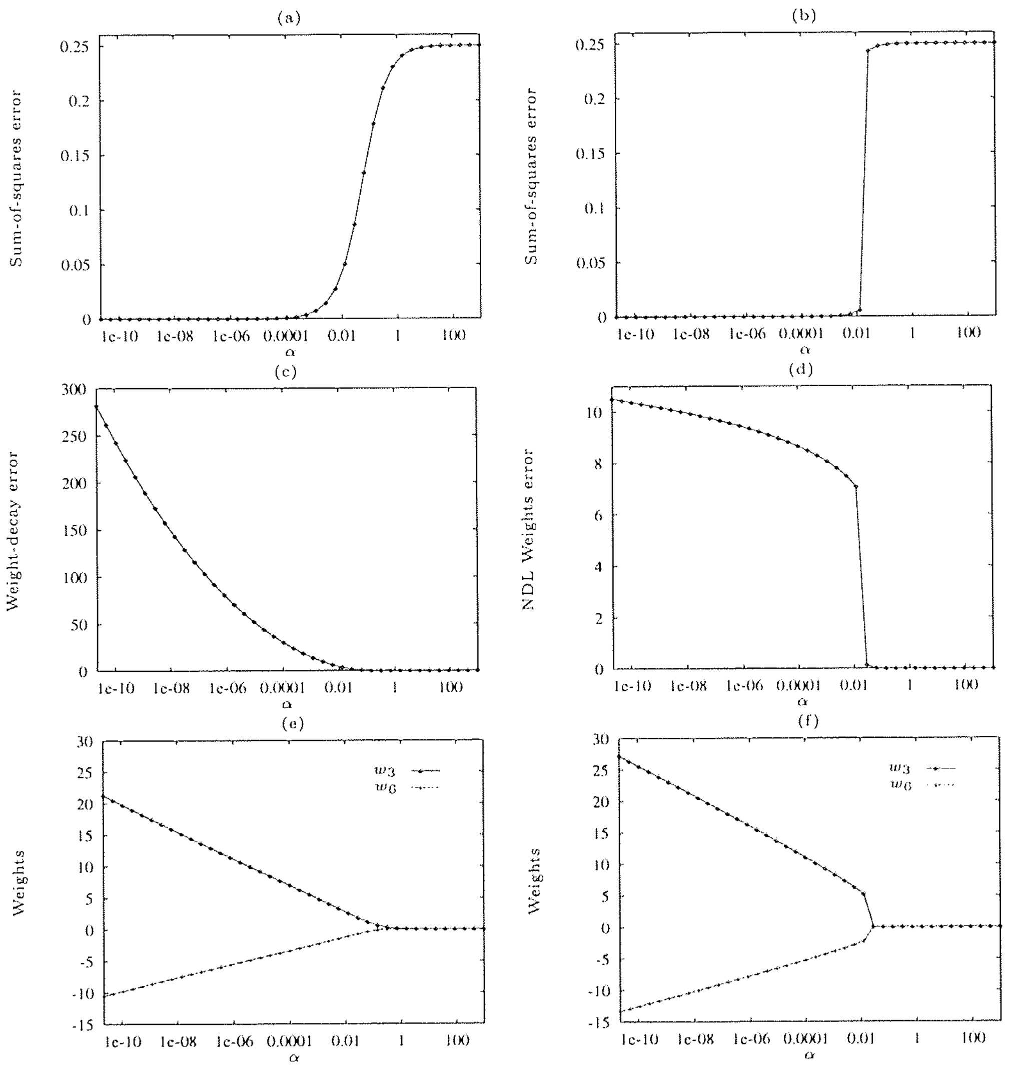

Figure 8.2: The sum-of-squares data and network error for the two pattern training set. Training was done with conventional weight-decay and NDL weights with weight $\epsilon = 0.1$. (a), (b) The training data error. (c), (d) The network error. (e), (f) The values of two of the weights in the network. Weight w_6 corresponds to the bias in the problem.

Network	w_1	w_2	w_3	w_3	w_5	w_6
Sum-of-squares	0.24	0.11	21.26	0.37	0.05	-10.53
NDL, $\epsilon = 1$	-6.22e-16	-8.07e-17	27.18	-1.18e-16	-4.56e-16	-13.34
NDL, $\epsilon = 0.1$	-2.17e-18	-4.40e-19	27.19	3.85e-18	-7.15e-20	-13.32
NDL, $\epsilon = 0.05$	-2.59e-20	1.63e-18	27.23	8.23e-19	-3.43e-19	-13.33
NDL, $\epsilon = 0.01$	-3.15e-21	2.62e-21	27.08	-1.29e-21	1.28e-21	-13.30

Table 8.1: The values of the individual weights in the network for different weight ϵ.

constrained. The ideal solution would be to constrain as many of the weights to zero as possible. This would leave only a few weights with large values capable of modelling the data. By using NDL regularisation large weights can exist with very small ones.

Figure 8.2 shows results for training on the above two patterns with the weight-decay and NDL weights regulariser. The sum-of-squares cost function was used for both methods of regularisation. Figure 8.2a shows the training sum-of-squares data error for each value of the regularisation parameter α. The α-survey is quite smooth and the error indicates a network output of (0.5, 0.5) for the two patterns when α is large and (1,0) when α is small. When the NDL weights regulariser is used the sum-of-squares data error shows a very clear 'cliff' in Figure 8.2b. For large values of α the weights are all driven to small values and again the network outputs are (0.5, 0.5). When α decreases to a value of 0.01 the information available within the network is enough for the 3rd input component to be modeled properly.

Figure 8.2c shows the training regularisation or network error for a network trained with the weight-decay regulariser before being multiplied by the regularising parameter α. The network error rises smoothly as the amount of regularisation decreases. In Figure 8.2d training with NDL weights showed a clear 'cliff' at an α value of 0.013. This is where the network was now flexible enough to model the data.

Figure 8.2e and 8.2f show the value of two of the weights in the network. The first weight corresponds to the weight where the input can have the two values 1 and 0 shown as w_3. The other weight shows the bias shown as w_6. As α decreases these two weights change smoothly for weight-decay regularisation. However, the other weights in the network do not approach zero. Table 8.1 shows the weight values for different training runs for weight-decay and NDL weights, when the regularisation parameter α approaches 0 in the α-survey. Figure 8.2f shows how the value of the third weight changes rapidly to model the 3rd input component. The bias term also changes in this example for NDL weights regularisation. By looking at the value of the other weights in Table 8.1 we see that their values are negligible. This shows a large decrease in the

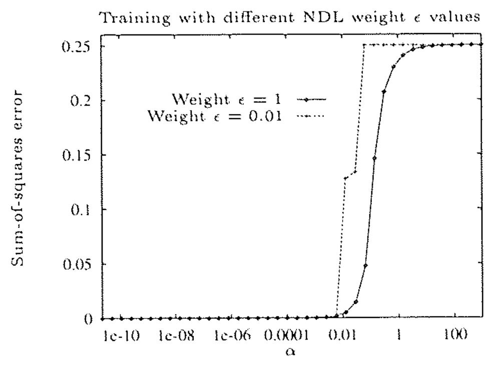

Figure 8.3: The sum-of-squares data error for the two pattern training set. Networks were trained with weight $\epsilon = 1$ and weight $\epsilon = 0.01$.

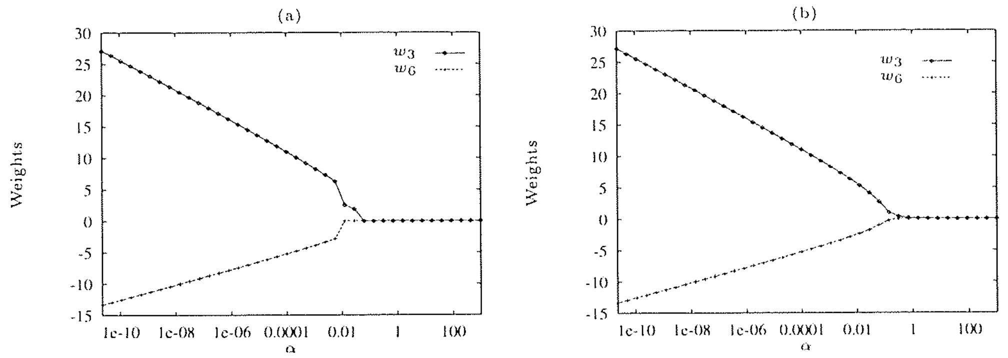

Figure 8.4: The change in value of two of the weights in the network as α is decreased for the two pattern training set. (a) A network where weight $\epsilon = 0.01$. (b) A network where weight $\epsilon = 1$.

amount of information needed to code this particular mapping.

In the example shown in Figure 8.2 a weight ϵ value of 0.05 was used. Figures 8.3 and 8.4 show similar results for networks trained with different weight ϵ's of 1 and 0.01. When ϵ was 1 the training α-survey looked similar to training with standard weight-decay. With a weight ϵ of 0.01 two steps occurred in the training α-survey. Looking again at the values of the network's most significant weights shows sudden increases in the values of the individual weights. At the first step the bias w_6 remains 0, however the weight value of w_3 is not large enough to cause the sigmoid to saturate and give a desired value of 1.

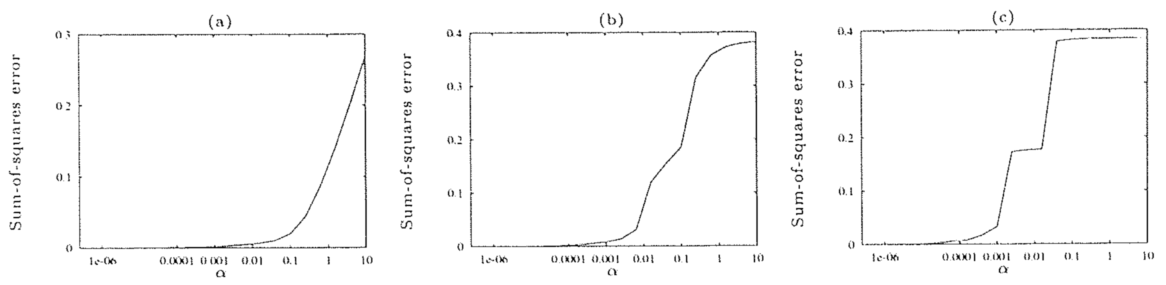

Figure 8.5: The data error for networks trained with the sum-of-squares cost function and different regularisers on the noisy parabola training set. (a) Weight-decay. (b) NDL weights, weight $\epsilon = 0.1$. (c) NDL weights, weight $\epsilon = 0.01$.

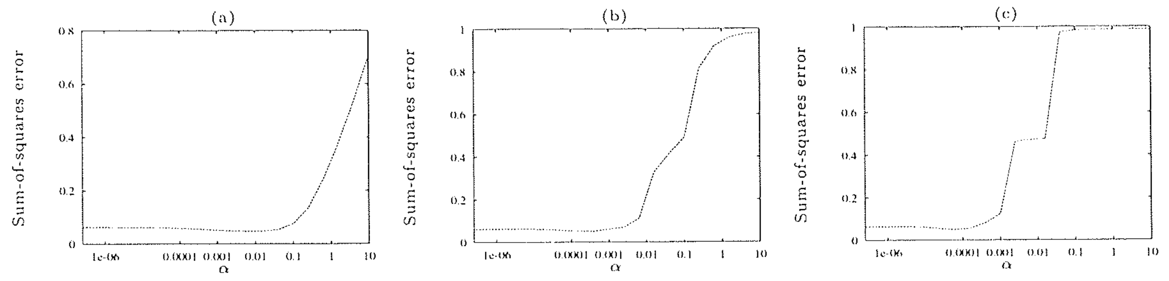

Figure 8.6: The data error for networks trained with the sum-of-squares cost function and different regularisers on the noisy parabola test set. (a) Weight-decay, minimum error $E_{Data} = 0.0466$ and $\alpha = 0.00631$. (b) NDL weights, weight $\epsilon = 0.1$, minimum error $E_{Data} = 0.0506$ and $\alpha = 0.000158$. (c) NDL weights, weight $\epsilon = 0.01$, minimum error $E_{Data} = 0.0487$ and $\alpha = 6.310e - 05$.

8.2 Using NDL weights for the parabola problem

The last section showed an example of how the NDL weights regulariser could give zero weights. In this example regularisation was not necessary to show the steps in the training α-survey as the weights could have been chosen by simple inspection of the data. The NDL weights regularisation cost function was used to train a network on the noisy parabola data set. The sum-of-squares cost function was used for the data. Again an RBF network was used with 60 basis functions, one for each input, and the α-survey generated. All training was again done to completion.

Figure 8.5 shows the effect of using different values of weight ϵ. The training α-survey looks similar to sum-of-squares training when weight ϵ is large in Figure 8.5b. As the value of weight ϵ decreases, steps can be seen in the α-survey. In Figure 8.5c there are two steps in the α-survey. The top plateau corresponds to the network giving a constant output for all inputs. The second plateau fits a straight line to part of the data. Finally the network fits most of the data for values of α smaller than 0.001.

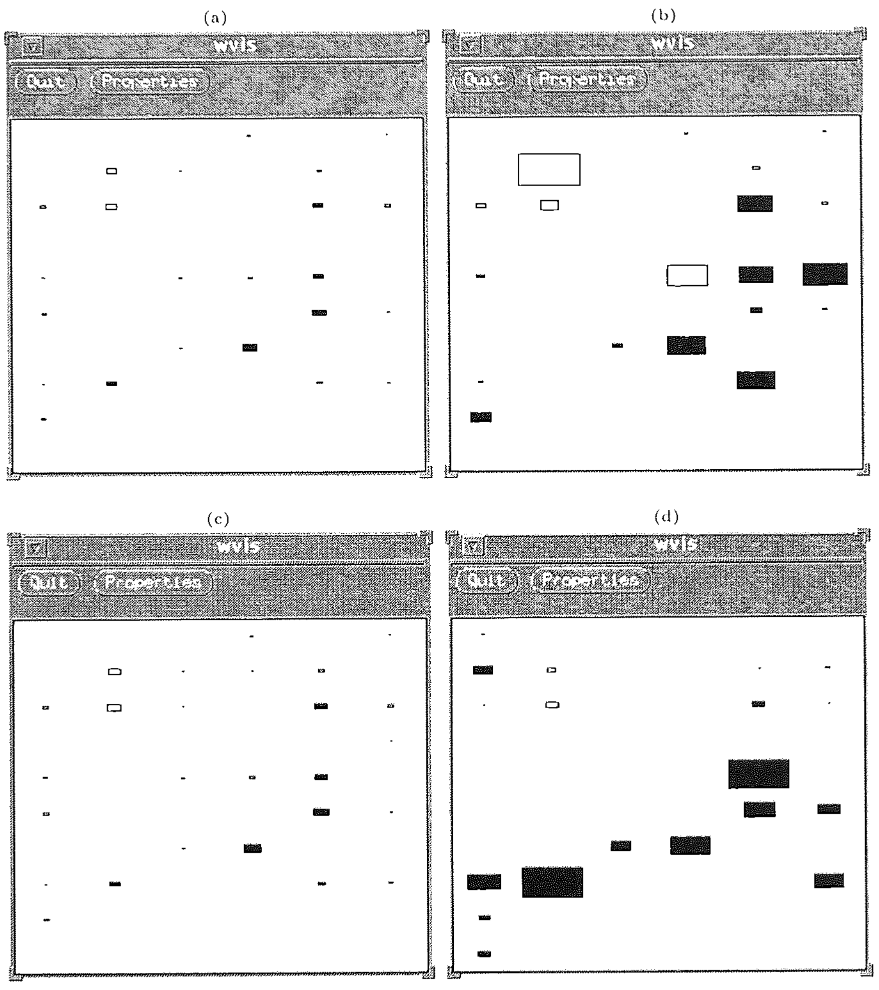

Figure 8.7: The weights of a network used to interpolate a simple parabola. Blanks represent zero valued weights. Networks giving a test set data error minimum were used. (a), (c) The weights when conventional weight-decay is used with 37 and 34 blank weights respectively. (b), (d) The NDL weights for weight ϵ set at 0.1 and 0.01 with 41 blank weights each.

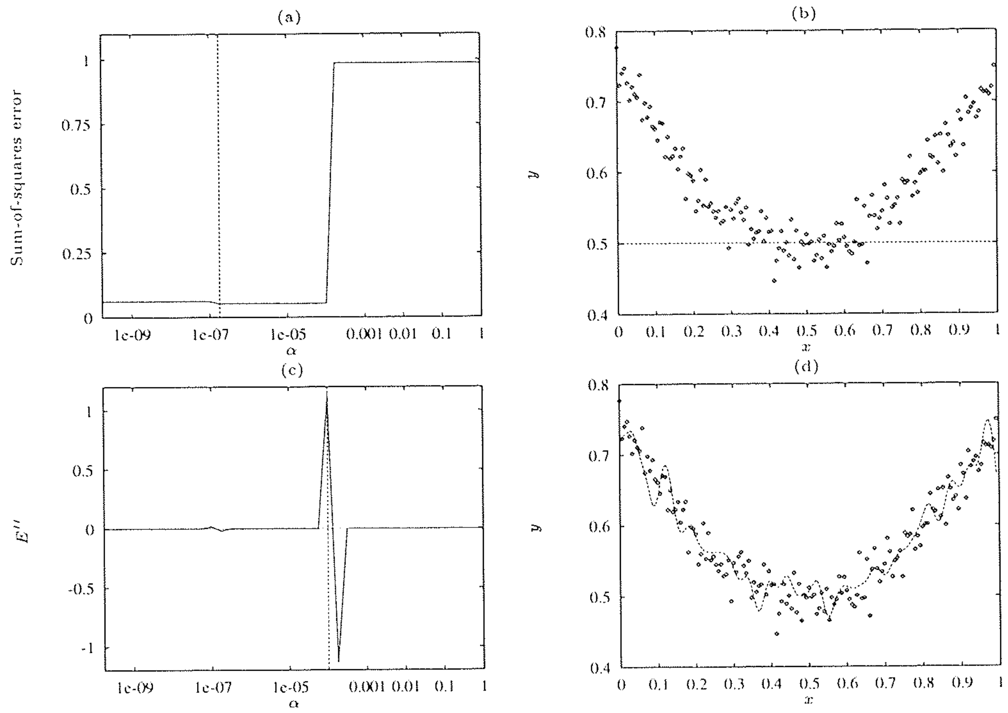

Figure 8.8: Training a network with a very small value weight ϵ of 0.00001 on the noisy parabola training set. (a) The test set errors with a minimum error of 0.0528 at an α value of 1.778e-07. This is shown by the vertical line. (c) The second derivative of the training error α-survey. The maximum is shown by the vertical line. (b), (d) The network outputs for the test set corresponding to the plateaus in the training α-survey at α values of 0.00316 and $5.623e-07$.

Figure 8.7 shows the weights from the basis function outputs to the network output as a 10 by 6 matrix. The weights were chosen corresponding to the networks that gave the minimum test set error for both weight-decay and NDL weights where weight ϵ was 0.1 and 0.01. The test set α-surveys are shown in Figure 8.6. Figure 8.7 shows a diagram first used by Hinton, to show the relative values of the weights. Large black boxes denote positive weights. The empty boxes denote negative weights. The weights for weight-decay regularisation are shown twice so that the proper scaling can be done with the weights trained with NDL regularisation. These diagrams do not show how the weight values change as shown before. They do show that for weight-decay regularisation the weights have similar values relative to each other. When the network was trained with weight ϵ of 0.1, a number of weights have been allowed to have large values. For weight-decay the number of zero weights was 37 compared to 41 for NDL weights. When the value of weight ϵ is 0.01 a relatively larger number of weights have reached zero values. In this case 41 weights compared to 34 with weight-decay regularisation.

In Section 4.2 it was shown how the NDL training α-survey could be used to set the regularisation parameter α. Figure 8.8a shows some results from training a network using NDL weights where weight ϵ is set to a very small value and the sum-of-squares cost function was used for the data. In Section 4.5.1 it was shown that the smaller the value of ϵ in NDL data training the better the second derivative was at indicating a value to set α. A small value of weight ϵ does cause the 'cliff' to be sharper. It does not in this case point to the minimum test set data error shown in Figure 8.8a, though it is very near to the minimum.

Figure 8.8b and 8.8d also show the network outputs at the two plateaus in the testing α-survey. The top plateau corresponds to networks that produce the same output for all the inputs in Figure 8.8b. When α has a small value the network fits the points more closely just before the 'cliff' in Figure 8.8a.

Training with NDL in the weights also allows α-surveys to be generated more quickly. By using RBF networks in many of the experiments the networks were allowed to train until no further change in the error was achievable. For weight-decay regularisation each decrease in the value of α led to networks that had to train for many iterations before they would converge. By using NDL in the weights there are particular points in the α-survey where the network is free to obtain a particular solution to the problem. The training error at these points decreases rapidly, in the same way that any MLP will have large changes in the error for the first few iterations in training. By using NDL in the weights the network outputs stay fairly constant for most parts of the survey. The majority of training occurs over a very short region of α. This makes the whole training and generation of α-surveys less computationally expensive.

8.3 Using data NDL and network NDL

In Section 3.1.1 a time series problem was described that followed an 'inverse' XOR pattern. The outputs of the series could be determined by examining only the previous two inputs. The most desirable pattern of weights would be one that only paid attention to the last two input components. This would minimise the information required to reconstruct the outputs of the time series from the inputs. It would be desirable if the weights feeding from previous time-steps were driven to zero. Then a pruning algorithm can be used to characterise the data with a smaller network and enhance the speed of training and testing. By using the NDL principle in the weights a simple example was shown in Section 8.1 describing how unnecessary inputs have their weights driven to zero. This could also be used in this problem. The main aim however of this section is to show how both NDL in the data and NDL in the weights can be used together.

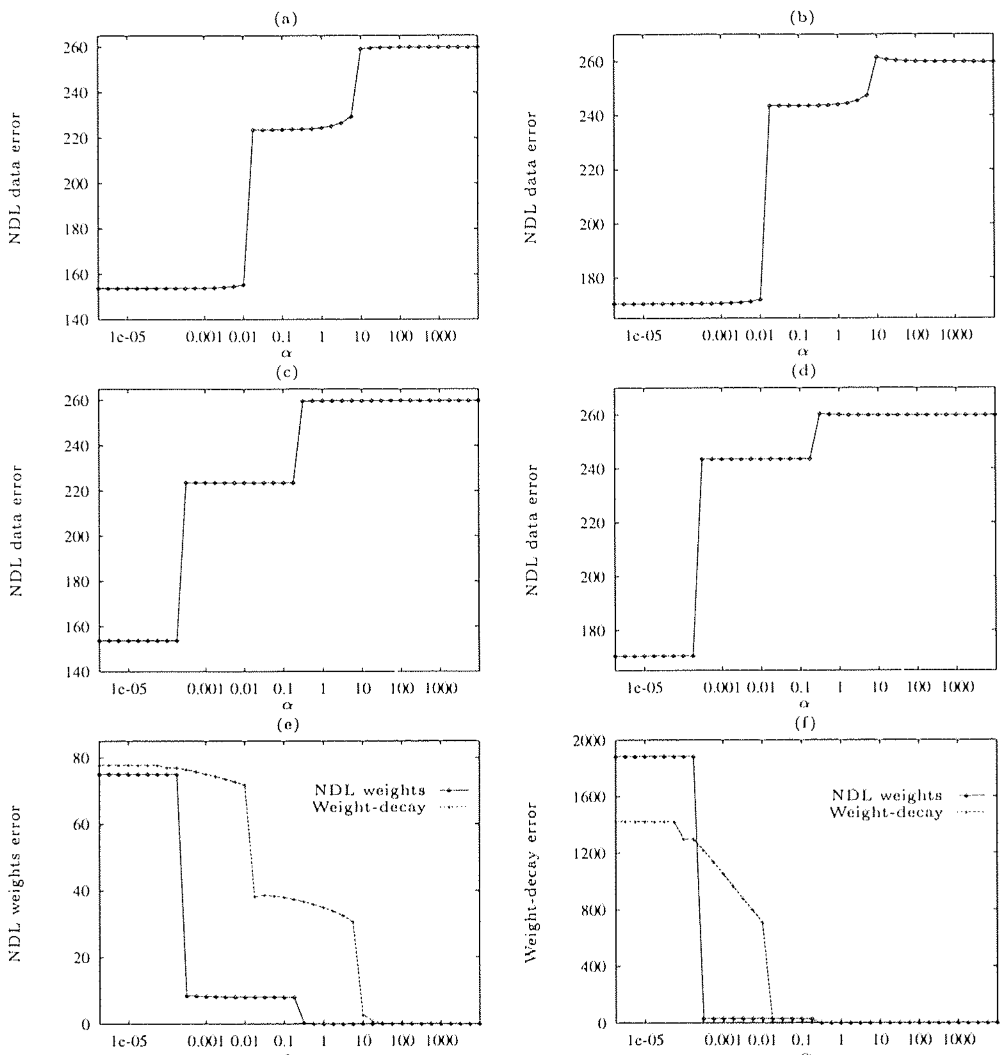

Figure 8.9: Training and testing on the 'inverse' XOR data set with weight-decay and NDL weights, weight $\epsilon = 0.1$. The NDL cost function is used for the data in both networks with $\epsilon = 0.05$. (a) The training NDL data error when weight-decay regularisation is used. (c) The training NDL data error for the data when NDL weights regularisation is used. (b) The test set NDL data error for a network trained using the weight-decay cost function. (d) The test set NDL data error for a network trained using the NDL weights regulariser. (e) The NDL weights test error for both networks trained with weight-decay and NDL weights. (f) The weight-decay error for both networks.

The 'inverse' XOR problem was described in Section 3.1.1. It was presented as a problem of extracting the deterministic part of the data from the non-deterministic part. Figure 8.9 shows a number of training and testing α-surveys. Figure 8.9a shows the NDL error for the data while training and Figure 8.9b the error for testing with a network trained using the NDL data cost function and the weight-decay regularisation cost function. The three plateaus shown in Figure 8.9b correspond to the network outputs of (0.5,0.5) for all patterns and then either of only two combinations of (0,1) and (1,0). Finally the lowest plateau gave all combinations of (0,1), (1,0), (1,1) and (0,0), following the 'inverse' XOR. In these two graphs the 'cliffs' are clearly seen. However it is also noticeable that there is a slight variation in the error just before and after the 'cliff', for example when α is around 10 on Figure 8.9a. The same effects are noticeable for testing.

Figure 8.9c and 8.9d show the same α-surveys with NDL used in the weights. The slight changes in the error are not noticeable in these plots. The NDL in the weights has served to exaggerate the effect of the sudden increase or decrease in the error of the data. It shows more clearly the points in the network where the weights, which contain the information available, can accommodate the mapping being modeled.

The error contributed by the weights is shown in Figure 8.9e and 8.9f. Figure 8.9e shows the network error for the test set according to the NDL weights regulariser for the two networks. One trained with weight-decay and the other with NDL weights. The error for using weight-decay stays well above the error when using NDL weights. This is because more of the weights in the network have been driven to values within the scale of ϵ, which was 0.1 in this case. Figure 8.9f shows the same trained networks with the errors calculated using the weight-decay regulariser and show the opposite effects. The NDL weight error is larger than weight-decay error for small α, but only while enough of the weights are unconstrained enough to model the data. At small values of α some of the weights have been allowed to have large values. Weight-decay training will produce a set of weights that are more similar. The error for weight-decay rises more rapidly, as α decreases, before the error for the network trained with NDL weights. This is because all the weights in the NDL trained network have been driven to values within the scale of ϵ. Only at particular points do some of the weights gain large values, larger than the weight-decay trained network.

8.3.1 NDL weights and regression problems

In Section 4.5.1 it was shown how using NDL in training could aid in chosing the regularisation parameter α for a network. This was done to get good performance on a test set provided the value of ϵ was small enough and training would give a smooth

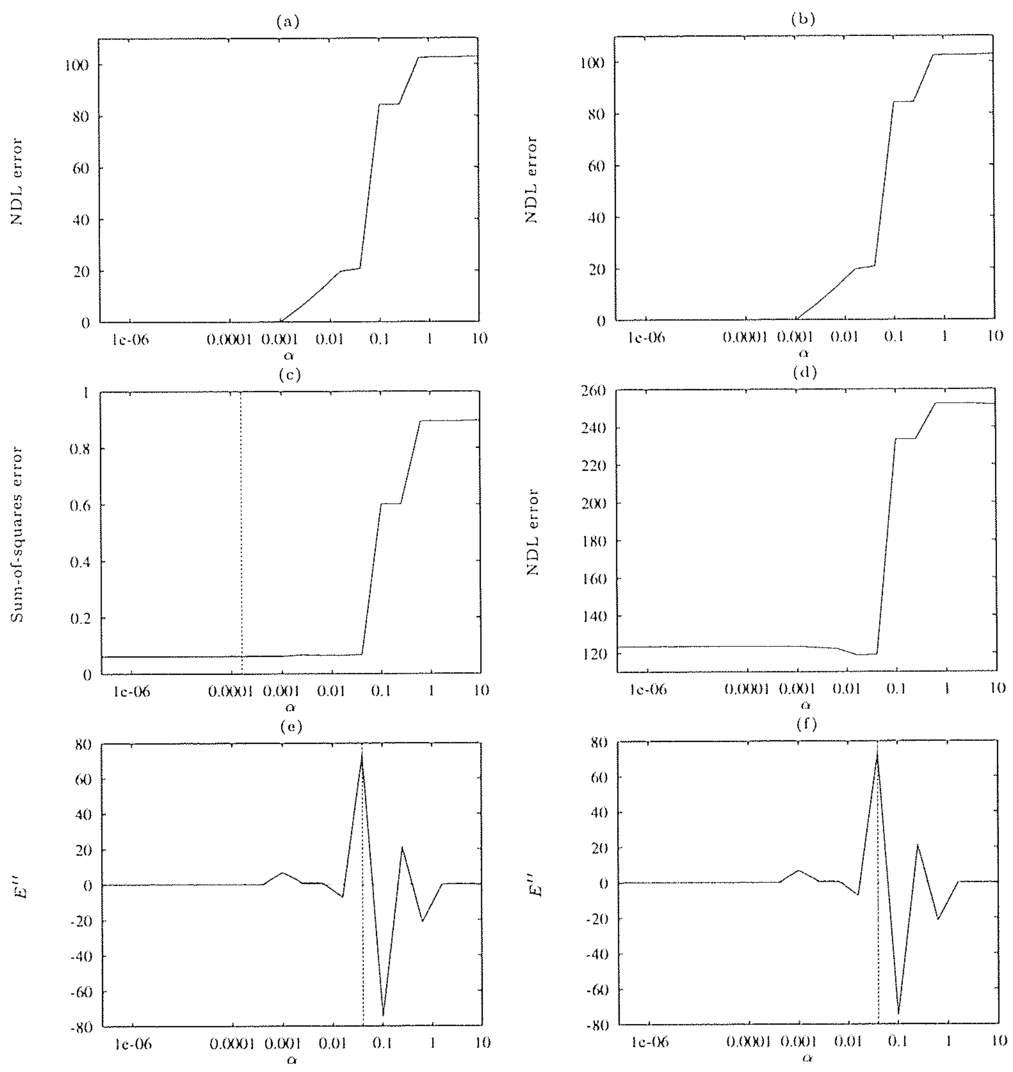

Figure 8.10: The results of training on the noisy parabola training set. The NDL data cost function with $\epsilon = 0.01$ and the NDL weights regulariser with weight $\epsilon = 0.001$ were used. (a), (b) The training NDL data error α-survey. These are the same so that they can be compared with the test set error. (c) The test set sum-of-squares data error. (d) The test set NDL data error. (e), (f) The second derivative of the training NDL data error.

training α-survey. It was shown in a previous section that by using NDL in the weights the training α-survey could be sharpened up with more prominent 'cliffs'. Training was repeated for the noisy parabola problem using NDL in the weights. The weight ϵ was chosen to be 0.001 and ϵ for NDL over the data was set at 0.01.

Figure 8.10 shows the results of training with both the NDL cost function and NDL weights regulariser. Using NDL weights has sharpened up the training α-survey. However, the second derivative of the training NDL data error does not point to the minimum test set error measured using the sum-of-squares cost function in Figure 8.10c. The training α-survey and second derivative graphs were shown again in Figure 8.10b and 8.10f. This time they are compared to testing using the NDL cost function. In this case the second derivative did point exactly to the minimum test set error. Fitting the most points at this minimum does not guarantee that the sum-of-squares will be at a minimum. Though the second derivative did not point to the minimum it did point to an error that was very close to it.

8.4 Summary

NDL was used to give a description length interpretation to the weights. A simple problem with redundant inputs was used to show that NDL in the weights reduced the value of the weights from redundant inputs to zero. Using NDL weights instead of a weight-decay term showed a 'cliff' in the training α-survey using the sum-of-squares cost function. Several values for weight ϵ were tested in the noisy parabola problem and showed some levels of structure. Hinton diagrams showed that the NDL weights regulariser showed larger weights for the minimum test error network and a larger number of weights that were driven to zero. The second derivative calculation was taken of the sum-of-squares cost function training α-survey with NDL weights. This did not point to the minimum test set error. Using weights NDL in conjunction with data NDL did have the effect of showing more clearly defined 'cliffs' in the α-survey for the 'inverse' XOR time series problem. Finally both weights NDL and data NDL were used in the noisy parabola problem, again having the effect of showing more clearly defined 'cliffs' in the α-survey. It was shown in the last section that using NDL in the weights instead of the conventional weight-decay term reduced the time of training a series of networks to form an α-survey. This was because of the presence of the 'cliff' where training occurs only over a small range of α.

Chapter 9

Conclusions

The NDL cost function can approximate the mode of a given set of data instead of the mean. This is because models that fit points within the ϵ scale are preferred to fitting points outside the ϵ scale. The choice of ϵ affects the choice of a mode in some data during unsupervised learning. The application of the NDL cost function on the Inverse Kinematics of a Robot arm gave good results. The network chose the elbow down in some regions and the elbow up in the remaining regions. The results were particularly good when RBF networks were used where the RMS error achieved was very low. The NDL cost function worked with toy problems with multi-modal features in the data set. It was particularly good at solving multi-valued problems with a clear separation between the clusters.

RBF networks proved to be a very fast method of learning. Other methods, such as the Mixture Density Network of Bishop solve these types of problems by generating the conditional probability density of the targets given the inputs. The use of the NDL cost function in backpropagation has no need to build up a model of the conditional probability density. However the NDL cost function can have problems with some problems that contain discontinuous regions.

The dependence of the error to regularisation shows 'cliffs' indicating levels of structure when using the NDL cost function. To get networks that show different levels of structure, α must be set to a value corresponding to the base of the 'cliffs' in the training α-survey. These represent the most regulated networks at different levels of structure. The dependence of the error to regularisation can be shown by generating an α-survey. The second derivative of the training α-survey shows more accurately the base of the 'cliffs' in the α-survey. However, other techniques such as curve fitting using polynomials could be used.

Second derivative peaks of the training α-survey point to good values to set the regularisation parameter α on some problems. For simple problems where the structure of the data is known the second derivative peaks do point to the minimum test errors.

This is becuase the most appropriate cost function to test the network can be chosen. Where the structure of the data is not known a particular testing cost function may not coincide with the second derivative peak.

The NDL cost function can be used to generate training α-surveys on classification problems. As with regression problems the second derivative of the training α-survey can indicate structure present in the data. NDL cost function training works well for problems with no overlapping classes. The Cross-Entropy cost function performs better on problems where the classes overlap.

The use of NDL cost function training chooses the most probable target in time series prediction. For simple problems where there is a multi-modal feature in the data the predictions can be very accurate after many iterated predictions. The NDL cost function extracts the deterministic part of the data from data that contains a random part and a deterministic part. However, the possibility of large errors when wrong predictions are made are greater. The mean NDL data error for a series of iterated predictions for NDL cost function training is lower than for sum-of-squares cost function trained networks for problems that contain a multi-modal feature. For problems were there is no multi-valued feature there is no advantage in using the NDL cost function.

The number of time-steps used in the network inputs is very important for time series prediction. It is useful for determining when particular paths are taken in the time series. A multi-modal feature in some problems can be followed with relatively few time-steps. This has the advantage that the networks take less time to train. Some data sets may have a multi-modal feature that exists over many time-steps. Training networks for these problems will take more time and performance will degrade as the test sets become smaller.

Small values of ϵ are preferable to large ones for showing structure at a large and small scale. However, small values of ϵ can lead to numerical complications and α-surveys that do not look smooth. This lead to the development of the Adaptive NDL method. Where the 'separation' between the possible modal values is very small it is preferable to use the Adaptive NDL method.

It was found in Section 4.6 that small values of ϵ were preferable to large ϵ for showing structure at a large and small scale. Numerical complications for small ϵ lead to the development of the Adaptive NDL method. This gave better results for problems where the 'separation' between the different multi-valued points was small.

The NDL cost function can give a description length interpretation to the weights. Like the weight elimination method of Weigend [89], weights that play no part in the representation of the data are driven to zero. The NDL weights regulariser works harder to eliminate weights giving a large negative cost to weights that approach 0.

Training α-surveys using the sum-of-squares cost function and NDL weights regularisation show 'cliffs' similar to NDL cost function training. However this technique does not help in the selection of α as NDL cost function training does. The NDL cost function and NDL weights regularisation make the 'cliffs' more pronounced, which may help the selection of the regularisation parameter in some problems.

The computational cost of generating α-surveys is a drawback to selecting a value for the α parameter. Many networks are trained over a large range of α to get a good survey. The number of values used in the survey must be large enough so that the base of the 'cliffs' can be accurately found. This problem was partially solved by using the weights of one network to initialise the next. The use of the NDL weights regulariser also caused most of the training to occur at particular points.

9.1 Future work

Most of the work presented here depended on the generation of α-surveys over a training set. As with many methods of training, problems of local-minima will seriously affect test errors at different values of α. More research could be undertaken to generate more accurate surveys. Changing α more slowly in regions where the error changes the most rapidly could give more accurate results in setting regularisation parameters.

For the robot arm problem and the inverse sine problem the Neural Networks, which model continuous functions, gave large errors around the boundaries of the discontinuous regions of the problem. More research could be directed into identifying these regions and perhaps training multiple networks on each region.

For experiments carried out on the noisy two-lines problem the training and testing showed different results when training was performed with both the sum-of-squares and NDL cost function. The minimum test set data error was very different for both cost functions. This was seen as an indicator of a multi-modal feature in the data. For some problems, training with the NDL cost function did not appear to give a clear underfitting and overfitting performance when tested. This was due to the way the cost functions used to test the networks interpreted the errors. When the data set was known not to be multi-modal then the sum-of-squares cost function was used, for example in the noisy parabola problem. If the nature of the problem is not known then the choice of testing cost function will give quite different results. More research could be done in identifying which testing cost function is appropriate for a particular data set.

Some poor results were obtained for some of the classification problems and regression problems. A single bad result is enough to cast doubt on this method if it is to be used extensively for regularisation parameter selection. However, it is believed that with improved training techniques these peaks will point to the minimum test set data error. In some of the experiments multiple peaks pointed to different levels of structure in the data. The question of which peak to choose must be answered.

By using the NDL cost function a graph could be drawn where the error at one time-step could be plotted against the error at the next time-step. It was shown in a the tri-valued time series problem. This showed clusters of points indicating that if the error was small at one point then the likelihood of the next error being small would be high. More research could be done to quantify this likelihood and how this information could be used in a system making predictions on this type of data set.

In Section 6.3.2 it was stated that an ensemble of networks could be trained on each mode in the data. In this way the outputs of each of the networks could be inspected to see which one gave the smallest error. It was seen in Section 6.2 that small errors at one time-step were likely to give small errors in the next time-step. More research could be done into training networks on multiple modes and deciding which network in the ensemble to use.

In some of the experiments, for example the Auto-Price data set, multiple peaks were seen. The largest size peak did not always point to the minimum error. In some cases the peaks did not point to any notable increase in test set performance. It was not understood fully why this occurred. It was also not fully understood why the performance of NDL cost function training grew worse with networks trained with smaller values of ϵ in the classification problems.

More research could be undertaken to determine the correct value of ϵ for a given problem. The unsupervised Gaussian mixture problem highlighted that different values of ϵ caused a network to converge on different clusters. For the regression problems it was noted that a small value of ϵ would indicate small and large scale structure in the data. More research could be conducted to look into the affect of ϵ on unsupervised problems and the characteristics of the α-survey.

More research could be done into improving network convergence when ϵ is very small. This was found in many of the regression and classification problems. Small values of ϵ resulted in α-surveys that were not smooth, giving multiple peaks in the second derivative of the error. This problem was solved partly by using Adaptive NDL. More research could be done into looking into the Adaptive NDL procedure.

The NDL cost function was applied to the Lorenz Attractor and Bach's last Fugue. Because of the computationally expensive generation of α-surveys only part of the data sets were used. The number of time-steps in the inputs were also restricted for this reason. For example in the Bach's last Fugue data set only 1 and 2 time-steps were used. The Lorenz Attractor orbits never repeat and hence the network had to generalise well from training on a relatively small training set. The performance of NDL cost function trained networks for larger data sets would provide a better understanding of how NDL would perform for other deterministic and chaotic data generators.

Appendix A

Mathematical details

A.1 Applying the NDL learning rule to Backpropagation

In order to perform backpropagation the derivative of the error with respect to the weights must be found. The following shows that derivation and an example pseudocode fragment based on the C programming language.

$$E = \sum_p \ln \left(\frac{\sum_i e_{ip}^2 + \epsilon^2}{\epsilon^2} \right)^{\frac{1}{2}} \tag{A.1}$$

$$e_{ip} = f(\mathrm{w}, x_{ip}) - t_{ip} \tag{A.2}$$

where f is the function implemented by a Neural Network for node i in pattern p.

$$\begin{aligned}
\frac{dE}{d\mathrm{w}} &= \sum_p \frac{1}{\left(\frac{\sum_i e_{ip}^2 + \epsilon^2}{\epsilon^2}\right)^{\frac{1}{2}}} \frac{d}{d\mathrm{w}} \left(\frac{\sum_j e_{jp}^2 + \epsilon^2}{\epsilon^2}\right)^{\frac{1}{2}} \\
&= \sum_p \frac{1}{\left(\frac{\sum_i e_{ip}^2 + \epsilon^2}{\epsilon^2}\right)^{\frac{1}{2}}} \frac{1}{2\left(\frac{\sum_j e_{jp}^2 + \epsilon^2}{\epsilon^2}\right)^{\frac{1}{2}}} \frac{d}{d\mathrm{w}} \left(\frac{\sum_j e_{jp}^2 + \epsilon^2}{\epsilon^2}\right) \\
&= \sum_p \frac{1}{2\left(\frac{\sum_j e_{jp}^2 + \epsilon^2}{\epsilon^2}\right)} \sum_j \frac{2e_{jp}}{\epsilon^2} \frac{de_{jp}}{d\mathrm{w}} \\
&= \sum p \frac{1}{2} \sum_j \frac{2e_{jp}}{\epsilon^2} \left(\frac{\epsilon^2}{\sum_k e_{kp}^2 + \epsilon^2}\right) \frac{de_{jp}}{d\mathrm{w}} \\
&= \sum_{ip} \frac{e_{ip}}{E_p} \frac{df(\mathrm{w}, x_{ip})}{d\mathrm{w}}
\end{aligned} \tag{A.3}$$

where,

$$E_p = \sum_i e_{ip}^2 + \epsilon^2 . \tag{A.4}$$

A.1.1 Sample code for NDL error and gradient calculation

```
 .

 .

Sum-of-Error = 0.0;
for ( i = 1 to Number-of-outputs ) {
   Error = Target[i] - Output[i];
   Sum-of-error = Sum-of-error + (Error * Error);
}
Sum-of-error = Sum-of-error + (Epsilon * Epsilon);

for ( i = 1 to Number-of-outputs ) {
   Error = Target[i] - Output[i];
   Error-Gradient[i] = Error / Sum-of-Error;
}

Error = log(Sum-of-error / (Epsilon * Epsilon))));
 .

 .
```

A.1.2 Sample code for NDL well-fit points calculation

```
.

.


Well-fit-points = 0.0;

for ( i = 1 to Number-of-patterns) {
   Sum-of-error = 0.0;
   for ( j = 1 to Number-of-outputs) {
      Error = Target[j] - Output[j];
      Sum-of-error = Sum-of-error + (Error * Error);
   }
   Well-fit-points = Well-fit-points +
         log(sqrt(Sum-of-error + (Epsilon * Epsilon))) /
                   log(Epsilon);
}
.

.
```

A.2 NDL weights

For the NDL in the weights cost function the derivative of the error due to the weights with respect to the individual weights must be calculated:

$$E = \sum_i \ln\left(\tfrac{w_i^2+\epsilon^2}{\epsilon^2} \right) , \tag{A.5}$$

$$\begin{aligned} \frac{dE}{dw_i} &= \frac{1}{\frac{w_{ip}^2+\epsilon^2}{\epsilon^2}} \frac{dE}{dw_i} \frac{w_i^2+\epsilon^2}{\epsilon^2} \\ &= \frac{1}{\frac{w_i^2+\epsilon^2}{\epsilon^2}} \frac{2w_i}{\epsilon^2} \\ &= \frac{2w_i}{\epsilon^2} \frac{\epsilon^2}{w_i^2 + \epsilon^2} \\ &= \frac{2w_i}{w_i^2 + \epsilon^2} . \end{aligned} \tag{A.6}$$

Starting from,

$$E = \frac{1}{2} \sum_i \ln\left(\tfrac{w_i^2+\epsilon^2}{\epsilon^2} \right) , \tag{A.7}$$

$$\frac{dE}{dw_i} = \frac{w_i}{w_i^2 + \epsilon^2} \, . \tag{A.8}$$

A.2.1 Sample code for NDL weights gradient calculation

```
.
.
for ( i = 1 to Number-of-weights ) {
   Error-Gradient[i] = -Alpha *
                        (Weight[i] /
      ((Weight[i] * Weight[i]) + (Epsilon * Epsilon)));
}
.
.
```

A.3 Sample code for NDL well-eliminated weights calculation

```
.
.
for ( i = 1 to Number-of-weights) {
   Well-fit-points = Well-fit-points +
         log(sqrt(Weight[i] * Weight[i] + (Epsilon * Epsilon))) /
                  log(Epsilon);
}
.
.
```

A.4 Calculating the Cross-Entropy cost function gradient

In a *one-out-of-N* output coding scheme the target will have as many outputs as there are classes. For each class a single output will have a value of 1; all the others will have a value of 0. Each output will converge to the probability of any particular target node i being 1. The outputs of a network when added will not usually equal 1 so they must be normalised.

$$P(i \mid \mathrm{x}_p) = \frac{y_{ip}}{\sum_j y_{jp}} \tag{A.9}$$

The probability of an output matching the target for every pattern is:

$$p(D) = \prod_p \frac{y_{i,\text{correct } p}}{\sum_j y_{jp}} , \tag{A.10}$$

$$\begin{aligned} E = \log P(D) &= \sum_{ip} t_{ip} \log \frac{y_{ip}}{\sum_j y_{jp}} \\ &= \sum_{ip} t_{ip} \left(\log y_{ip} - \log \sum_j y_{jp} \right) . \end{aligned} \tag{A.11}$$

Taking the derivative of the error with respect to the weights gives:

$$\begin{aligned} \frac{dE}{dw} &= \sum_{ip} t_{ip} \left(\frac{1}{y_{ip}} \frac{y_{ip}}{dw} - \frac{1}{\sum_j y_{jp}} \sum_j \frac{y_{jp}}{dw} \right) \\ &= \sum_{ip} \frac{t_{ip}}{y_{ip}} \frac{dy_{ip}}{dw} - \sum_p \frac{\sum_j t_{jp}}{\sum_j y_{jp}} \sum_j \frac{dy_{jp}}{dw} . \end{aligned} \tag{A.12}$$

But since $\sum_j t_{jp} = 1$ we have,

$$\frac{dE}{dw} = \sum_{ip} \left(\frac{t_{ip}}{y_{ip}} - \frac{1}{\sum_j y_{jp}} \right) \frac{dy_{ip}}{dw} . \tag{A.13}$$

A.4.1 Sample code for Cross-Entropy error and gradient calculation

```
.
.
Sum = 0.0;
for ( i = 1 to Number-of-outputs ) {
   Sum = Sum + Output[i];
}

for ( i = 1 to Number-of-outputs ) {
   Error-Gradient[i] = ( Target[i] / Output[i] )-
                       (1.0 / Sum );
   Error = Error - Target[i];
}

Error = log(Sum-of-error / (Epsilon * Epsilon))));
.
.
```

A.5 Calculating the second derivative by finite differences

In Section 4.2 the data error was calculated for different values of the regularising parameter α. This was part of what was called an α-survey of training and testing. The following shows a method of finding the vector of second derivatives for y with respect to x:

$$w = \frac{-(y_i - y_{i-1})}{x_i - x_{i-1}} + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \tag{A.14}$$

$$t = \frac{-(x_{i-1} + x_i)}{2.0} + \frac{x_i + x_{i+1}}{2.0}, \tag{A.15}$$

$$y_i'' = \frac{w}{t}. \tag{A.16}$$

A.6 Choosing the range of values for α in the α-survey

The α-surveys consist of gradually decreasing values for the regularising parameter α in training a Neural Network. It was required that the second derivative of the error could be calculated. In order that the second derivative could be calculated for fixed intervals the logarithm of the value of α was used in all second derivative calculations by finite differences. Therefore the range of α had to be chosen so that the logarithms of these values would have fixed intervals. For example the two limits of 10 and 0.0001 might be chosen. The logarithm of each extreme was calculated. The range was divided by the number of intervals or steps, which were usually 20 or 40. The size of the division between each value was used to obtain the set of α values. Hence $d^2E(\alpha)/d\log(\alpha)^2$ can be estimated.

```
.
.
Range = log(Highest-Alpha) - log(Lowest-Alpha);
Division = Range / Steps;

for (i = 1 to Steps) {
   print( exp( log(Highest-Alpha) - (i * Division)) );
}
.
.
```

A.7 Setting up a Gaussian mixture for unsupervised training

For the data set used in Section 3.4.1 three Gaussian functions where used to create a mixture of probability densities.

$$A = \frac{1}{3}\sum_{i=1}^{3}\int_{-\infty}^{g}\frac{1}{\sqrt{2\pi}\,\sigma_i}e^{\frac{-(x-c_i)^2}{\sqrt{2}\,\sigma_i^2}}\,dx \qquad (A.17)$$

Equation A.17 shows that the area A is equal to the sum of the integrals of the 3 Gaussian functions, between $-\infty$ and g. The random number r which equals A is chosen from a uniform distribution between 0 and 1. The value of g is set at the mid-point of the two extremes representing $-\infty$ and $+\infty$. The area is calculated and compared to A. If the area is less then A, g is set to the mid-point of the present value and the lowest extreme. The extremes are altered to form a binary search.

Substituting $u = \frac{x-c_i}{\sqrt{2}\,\sigma_i}$,

$$\begin{aligned} A &= \frac{1}{3}\sum_{i=1}^{3}\int_{-\infty}^{\frac{g-c_i}{\sqrt{2}\sigma_i}} \frac{1}{\sqrt{2\pi}\,\sigma_i} e^{-u^2}\sqrt{2}\,\sigma_i du \\ &= \frac{1}{3}\sum_{i=1}^{3}\frac{1}{2}\int_{-\infty}^{\frac{g-c_i}{\sqrt{2}\sigma_i}} \frac{2}{\sqrt{\pi}} e^{-u^2} du \\ &= \frac{1}{3}\sum_{i=1}^{3}\frac{1}{2}\left[\int_{-\infty}^{0} \frac{2}{\sqrt{\pi}} e^{-u^2} du + \int_{0}^{\frac{g-c_i}{\sqrt{2}\sigma_i}} \frac{2}{\sqrt{\pi}} e^{-u^2} du\right] \\ &= \frac{1}{3}\sum_{i=1}^{3}\frac{1}{2}\left[\int_{-\infty}^{0} \frac{2}{\sqrt{\pi}} e^{-u^2} du + erf\left(\frac{g-c_i}{\sqrt{2}\,\sigma_i})\right)\right] . \end{aligned} \tag{A.18}$$

Where:

$$erf(x) = \frac{2}{\sqrt{\pi}}\int_{0}^{x} e^{-u^2} du \, , \tag{A.19}$$

$$\int_{-\infty}^{0} \frac{2}{\pi} e^{-u^2} = -\int_{0}^{-\infty} \frac{2}{\pi} e^{-u^2} = 1 \, . \tag{A.20}$$

So:

$$A = \frac{1}{3}\sum_{i=1}^{3}\frac{1}{2}\left[1 + erf(\frac{g-c_i}{\sqrt{2}\,\sigma_i})\right] . \tag{A.21}$$

The equations above show the calculations necessary for a mixture of 3 Gaussian functions. The general case of a mixture of any number of Gaussian functions is shown below where N_c is the number of Gaussian functions in the mixture.

$$A = \frac{1}{N_c}\sum_{i=1}^{N_c}\frac{1}{2}\left[1 + erf(\frac{g-c_i}{\sqrt{2}\,\sigma_i})\right] \tag{A.22}$$

A.8 Coarse-coding

Coarse-coding is a method of increasing the dimensionality of the input space so that a hyperplane can be found for the data more easily. It can be defined as a set of uniformly spaced one dimensional Radial Basis Functions. It is designed to be used in multi-layer networks. A number of intervals are chosen arbitrarily and from this the centres of $N-1$ Gaussian functions are calculated. The volume under each Gaussian depends on the values of the variances, but as these are the inputs and the targets are unaffected then no normalisation occurs. Figure A.1 shows coarse-coding with 4 Gaussian functions. For each value of an input vector x_i, 4 Gaussian functions are used to generate 4 new inputs to the Neural Network. For each function the integral is calculated to form part of the new vector of inputs:

$$x'_i = \int_{-\infty}^{x_i} e^{-(x-c_i/\sigma_i)^2} \, . \tag{A.23}$$

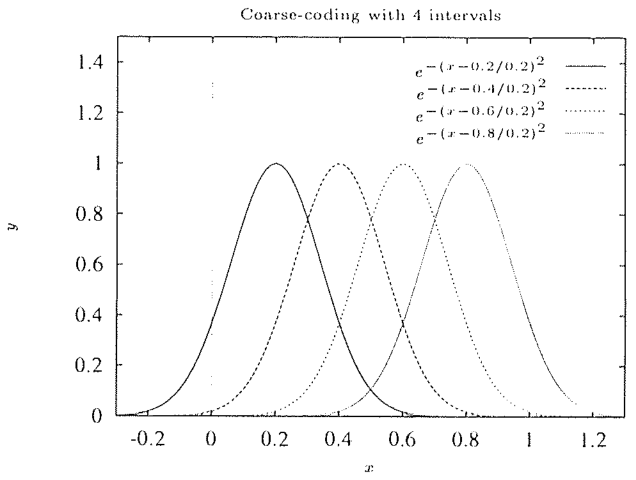

Figure A.1: Coarse-coding with 4 Gaussian functions with centres at fixed intervals in the range 0 and 1.

Bibliography

[1] Ash, T. Dynamimic node creation in Back-propagation Networks. Technical Report ICS Report 8901, Institute of Cognitive Science, University of California, San Diego, La Jolla, California 92093, USA, 1989.

[2] Baum, E.B., and Haussler, D. What size net gives valid Generalization. *Neural Computation*, 1:151–160, 1989.

[3] Becker, S. and Hinton, G.E. Spatial Coherence as an internal teacher for a Neural Network. Technical Report CRG-TR-89-7, University of Toronto, Toronto, 1989.

[4] Bellman, R.E. *Adaptive Control processes*. Princeton University Press: Princeton, NJ, 1961.

[5] Bishop, C.M. Curvature driven smoothing in Back-propagation Neural Networks. Technical Report CLM-P-880, AEA Technology, Culham Laboratory, Abingdon, UK, 1990.

[6] Bishop, C.M. Mixture Density Networks. Technical Report NCRG/4288, Aston University, Dept. of Computer Science and Applied Mathematics, Birmingham, UK, 1994.

[7] Bishop, C.M. Novelty detection and Neural Network validation. Technical Report NCRG/4289, Aston University, Dept. of Computer Science and Applied Mathematics, Birmingham, UK, 1994.

[8] Blumer, A., Ehrenfeucht, A., Haussler, D and Warmuth, M.K. Occam's Razor. *Information Processing Letters*, 24:377–380, 1987.

[9] Broomhead, D.S. and Lowe, D. Multivariable functional interpolation and Adaptive Networks. *Complex Systems*, 2, 1988.

[10] Chauvin, Y. A Back-propagating algorithm with optimal use of hidden units. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1 (Denver 1988)*, pages 519–526. San Mateo, CA, Morgan Kaufmann, 1989.

[11] Cybenko, G. Continuous valued Neural Networks with two hidden layers are sufficient. Technical report, Dept. of Computer Science, Tufts University, Medford, MA, 1988.

[12] Denker, J., Schwartz, D., Witner, B., Solla, S., Hopfield J., Howard, R. and Jackal, L. Automatic learning, rule extraction and Generalization. *Complex Systems*, 1987.

[13] Dirst, M. and Weigend, A.S. Baroque forecasting: On completing J.S. Bach's last Fugue. In *Time Series Prediction: Forecasting the future and understanding the past*, pages 151–155. Addison-Wesley, 1993.

[14] Falhman, S.E. and Lebiere, C. The Cascade-Correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 524–532. San Mateo, CA, Morgan Kaufmann, 1993.

[15] Frean, M. The Upstart Algorithm: A method for constructing and training feed-forward Neural Networks. *Neural Computation*, 2(2):198–209, 1990.

[16] Funahashi, K. On the approximate realization of continuous mappings by Neural Networks. *Neural Networks*, 2, 1989.

[17] Gallant, S.I. The Pocket Algorithm for Perceptron learning. Technical Report SG-85-20, Northeastern University College of Computer Science, USA, 1985.

[18] Gemen, S., Bienenstock, E. and Doursat, R. Neural Networks and the Bias-Variance Dilemna. *Neural Computation*, 4:1–58, 1992.

[19] Gray, R.M. Vector Quantization. *IEEE ASSP Magazine*, 4:4–28, 1984.

[20] Grossberg, S. Competitive Learning: From Interactive Activation to Adaptive Resonance. *Cognitive Science*, 11:23–63, 1987.

[21] Hansen, P.C. and O'Leary, D.P. Use of the L-curve in regularisation of discrete Ill-posed problems. *SIAM Journal on Scientific Computing*, 1993.

[22] Hanson, L.K. and Solamon, P. Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12:993–1002, 1990.

[23] Hanson, S.J. and Pratt, L. A comparison of different Biases for minimal network construction with Back-propagation. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 1 (Denver 1988)*, pages 177–185. San Mateo, CA, Morgan Kaufmann, 1989.

[24] Haykin, S. *Neural Networks.* Macmillan College Publishing Company, Inc, 1994.

[25] Hebb, D.O. *The Organisation of Behaviour.* Wiley, New York, 1949.

[26] Hergert, F., Finnoff, W. and Zimmerman, H.G. A comparison of Weight Elimination methods for reducing complexity in Neural Networks. In *Proceedings of the International-Joint Conference on Neural Networks (Baltimore, 1992)*, volume 3, pages 980–987. IEEE New York, 1992.

[27] Hertz, J., Krogh, A. and Palmer, R.G. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Reading, Mass., 1991.

[28] Hinton, G.E. Learning distributed representations of concepts. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society (Amherst 1986)*, pages 1–12. Erlbaum, Hillsdale, 1986.

[29] Hinton, G.E., Moody, J.E. and Hanson, S.J. Learning translation invariant recognition in a massively parallel network. In Goos, G., and Hartmanis, J., editor, *PARLE: Parallel Architectures and Languages, Europe Lecture Notes in Computer Science*, pages 1–13. Springer Verlag, Berlin, 1987.

[30] Hochreiter, S. and Schmidhuber, J. Flat minimum search finds simple nets. Technical Report FKI-200-94, Fakultät für Informatik, H2, Technische Universität Munchen, 80290 München, Germany, 1994.

[31] Hornik, K., Stinchcombe, M. and White, H. Multi-layer feedforward Networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[32] Huang W.Y. and Lippmann, R.P. Comparisons between Neural Nets and conventional classifiers. In *ICNN Proceedings, San Diego, California*, 21-24 June 1993.

[33] Hush, D.R. and Horne, B.G. Progress in supervised Neural Networks: What's new since Lippmann. *IEEE Signal Processing Magazine*, 10:8–39, 1993.

[34] Janssen, P., Stoica, P., Söderström, T. and Eykhoff, P. Model structure selection for multivariable systems by Cross-Validation. *International Journal of Control*, 47:1737–1758, 1988.

[35] Kirkpatrick, S., Gelatt, Jr., C.D. and Vecchi, M.P. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[36] Kohonen, T. *Self-organization and Associative Memory.* Berlin Springer, 1984.

[37] Kohonen, T. *Speech recognition based on Topology-Preserving Neural Maps.* MIT Press, Massachusetts, 1988.

[38] Krogh, A., and Hertz, J.A. A simple weight decay can improve Generalization. In Moody, J.E., Hanson, S.J. and Lippmann, R.P., editor, *Advances in Neural Information Processing Systems 4 (Denver 1991)*, pages 875–882. San Mateo, CA, Morgan Kaufmann, 1991.

[39] LeCun, Y. Une Procédure d'Apprentissage pour Réseau à seuil Assymétrique. *Cognitiva*, 85:599–604, 1985.

[40] LeCun, Y. Generalization and Network design strategies. Technical Report CRG-TR-89-4, Dept. of Computer Science, University of Toronto, 1989.

[41] LeCun, Y., Denker, J.S. and Solla, S.A. Optimal Brain Damage. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (Denver 1989)*, pages 598–605. San Mateo, CA, Morgan Kaufmann, 1990.

[42] Lee, Y. and Lippmann, R.P. Practical characteristics of Neural Networks and conventional pattern classifiers on artificial and speech problems. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 168–177. San Mateo, CA, Morgan Kaufmann, 1990.

[43] Linsker, R. Self-organsiation in a Perceptual Network. *Computer*, 21:105–107, 1988.

[44] Loredo, T.J. From Laplace to SuperNova SN 1987A: Bayesian Inference in Astrophysics. In P.F. Fougère, editor, *Maximum Entropy and Bayesian methods*, pages 81–142. Kluwer Academic Publishers, 1989.

[45] Lorenz, E. Deterministic non-periodic flow. *Journal of the Atmospheric Sciences*, 1963.

[46] Mackay, D.J.C. A practical Bayesian framework for Backpropagation Networks. *Neural Computation*, 4(3), 1992.

[47] Mackay, D.J.C. Bayesian Interpolation. *Neural Computation*, 4(3), 1992.

[48] Mackay, D.J.C. Hyperparameters: Opitimise or integrate out? In Heidbreder, G., editor, *Maximum Entropy and Bayesian Methods.* Dordrecht: Kluwer, 1994.

[49] McCulloch, W.S. and Pitts, W. A logical Calculus of ideas immanent in Nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[50] Mead, C.A. and Conway, L. *Introduction to VLSI systems.* Reading, MA: Addison-Wesley, 1980.

[51] Mezard, M. and Nadal, J.-P. Learning in feedforward layered Networks: The Tiling Algorithm. *Journal of Physics A*, 22:2191–2204, 1989.

[52] Michie, D., Spiegelhalter, D.J. and Taylor, C.C. (eds). *Machine Learning, Neural and statistical classification.* Prentice-Hall, 1994.

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

[54] Moody, J. and Darken, C.L. Fast learning in Networks of locally-tuned processing units. *Neural Computation*, 1:281–294, 1989.

[55] Morgan, N. and Bourlard, H. Generalization and parameter estimation in feed-forward nets: Some experiments. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2 (Denver 1989)*, pages 630–637. San Mateo, CA, Morgan Kaufmann, 1990.

[56] Morozov, V.A. *Regularization methods for Ill-posed problems.* Boca Raton, FL: CRC Press, 1993.

[57] Nowlan, S.J. and Hinton, G.E. Adaptive Soft Weight Tying using Gaussian Mixtures. In Moody, J.E., Hanson, S.J. and Lippmann, R.P., editor, *Advances in Neural Information Processing Systems 4*, pages 993–1000. San Mateo, CA, Morgan Kaufmann, 1992.

[58] Nowlan, S.J. and Hinton, G.E. Simplifying Neural Networks by Soft Weight-Sharing. *Neural Computation*, 4(4):473–493, 1992.

[59] Oja, E. *Subspace Methods of Pattern Recognition.* Letchworth, UK: Research Studies Press., 1983.

[60] Parker, D.B. Learning Logic. Technical Report TR-47, Center for Computational Research in Economics Management Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[61] Peterson, G.E. and Barney, H.L. Control methods used in a study of the Vowels. *Journal of the Acoustical Society of America*, 24(2):175–184, 1952.

[62] Plaut, D.S., Nowlan, S. and Hinton, G.E. Experiments on learning by Back-propagation. Technical Report CMU-CS-86-126, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1986.

[63] Poggio, T. and Girosi, F. Networks for approximation and learning. *Proceedings of the IEEE*, 78:1481–1497, 1990.

[64] Qazaz, C.S. Ph.D. First Year Qualifying Report. Dept. of Computer Science, Aston University, 1994.

[65] Quinlan, J.R. Combining Instance-Based and Model-Based learning. In Utgoff, P.E., editor, *Proc. ML'93*. San Mateo: Morgan Kaufmann, 1993.

[66] Rawlings, J.O. *Applied Regression Analysis*. Wadsworth and Brooks/Cole, Pacific Grove, CA, 1988.

[67] Reed, R. Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[68] Refenes, A.N. and Vithlani, S. Constructive learning by specialisation. In *Proceedings of the International Conference on Artificial Neural Networks, Helsinki, Finland*, June 1991.

[69] Richard, M.D. and Lippmann, R.P. Neural Network classifiers estimate Bayesian a posteriori probabilites. *Neural Computation*, 3:461–483, 1991.

[70] Rissanen, J. Minimum Description Length Principle. *Encyclopeadia of Statistical Sciences*, 5, 1985.

[71] Rohwer, R.J. and van der Rest, J.C. Minimum Description Length, Regularisation and Multi-Modal data. To appear in Neural Computation, 1995.

[72] Rosenblatt, F. *Psychological Review*, volume 65. 1958.

[73] Rosenblatt, F. *Principles of Neurodynamics*. Spartan Books, Washington DC, 1959.

[74] Rumelhart, D., Hinton, G.E. and Williams, R. Learning internal representations by Back-propagating errors. Technical Report ICS Report 8506, Institute of Cognitive Science, UCSD, 1985.

[75] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. Learning internal representations by error propagation. In Rumelhart, D.E., McClelland, J.L. and the PDP Research Group, editor, *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 1. MIT Press, Cambridge, MA, 1986.

[76] Scales, L.E. *Introduction to Mon-Linear Optimisation*. Macmillan, 1985.

[77] Schwartz, D.B., Samalan, V.K., Solla, S.A. and Deneker, J.S. Exhaustive Learning. *Neural Computation*, 2(3):374–385, 1990.

[78] Shannon, C.E. and Weaver, W. *The Mathematical Theory of Communication.* University of Illinois Press, 1949.

[79] Smith, M. *Neural Networks for Statistical Modeling.* New York: Van Nostrand Reinhold, 1993.

[80] J.V. Stone and C.J. Thornton. Can Artificial Neural Networks discover useful regularities. In *Fourth International Conference on Artificial Neural Networks Proceedings*, pages 201–205. IEE, 26-28 June 1995.

[81] Stone, M. Cross-Validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, B36:111–133, 1974.

[82] Stone, M. Cross-Validation: A Review. *Mathematisch Operationsforschung Statistischen*, 9:127–140, 1993.

[83] Sutton, R.S. Temporal Credit Assignment in Reinforcment Learning. Technical Report 84-02, University of Massachusetts, Computer and Information Sciences, Amherst, MA, 1984.

[84] Thodberg, H.H. Ace of Bayes: Application of Neural Networks with Pruning. Technical Report 1132E, The Danish Meat Research Institute, Maglegaardsvej 2, DK-4000, Roskilde, 1984.

[85] Tikhonov, A.N. and Arsenin, V.Y. *Solutions of Ill-posed problems.* Washington DC: W.H. Winston, 1977.

[86] Wang, C., Venkatesh, S.S., and Judd, J.S. Optimal stopping and effective machine complexity in learning. In *Advances in Neural Information Processing Systems 6 (Denver 1993)*, pages 303–310. Morgan Kaufmann, San Francisco, 1994.

[87] Wegner, T. and Peterson, M. *Fractal Creations.* Waite Group Press, 1991.

[88] Weigend, A.S., Huberman B.A. and Rumelhart, D.E. Predicting the Future: A Connectionist Approach. Technical Report 94305-2130, Stanford PDP Research Group, Stanford University, Stanford, California, 1990.

[89] Weigend, A.S., Rumelhart, D.E. and Huberman, B.A. Generalization by Weight-Elimination with application to Forecasting. In Lippmann, R.P., Moody, J.E. and Touretzky, D.S., editor, *Advances in Neural Information Processing Systems 3 (Denver 1990)*, pages 875–882. San Mateo, CA, Morgan Kaufmann, 1991.

[90] Werbos, P. Beyond Regression: New tools for prediction and analysis in the Behavioural Sciences. Ph.D. thesis, Harvard University, 1975.

[91] White, H. Learning in Artificial Neural Networks: A Statistical Approach. *Neural Networks*, 1:425–464, 1989.

[92] Widrow, B. and Hoff, M.E. *Adaptive Switching Circuits.* IRE WESCON Convention Record, New York, 1960.

[93] Wieland, A. and Leighton, R. Geometric Analysis of Neural Network capabilities. In *1st IEEE International Conference on Neural Networks, San Diego: CA*, volume 3, pages 385–392, 1987.

[94] Wolpert, D.H. On the use of evidence in Neural Networks. In Giles, C.L., Hanson, S.J. and Cowan, J.D., editor, *Advances in Neural Information Processing Systems 5*, pages 539–546. San Mateo, CA: Morgan Kaufmann, 1993.

[95] Wynne-Jones, M. Node splitting: A Constructive Algorithm for feed-forward Neural Networks. In Moody, J.E., Hanson, S.J., and Lippmann, R.P., editor, *Advances in Neural Information Processing Systems 2*, pages 1072–1079. Morgan Kaufmann, 1992.

[96] Zemel, R.S. A Minimum Description Length framework for Unsupervised Learning. Ph.D. thesis, Graduate Dept. of Computer Science. University of Toronto, 1993.