gallagher2000multi

Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling

Marcus Reginald Gallagher

B. Comp. Sc., Grad. Dip. Sc.

Department of Computer Science and Electrical Engineering

University of Queensland, St Lucia 4072, Australia

A thesis submitted for the degree of Doctor of Philosophy.

June 30, 1999 (Revised January, 2000)

Statement of Originality

To the best of my knowledge and belief, this thesis is original and my own work except as acknowledged in the text. Parts of this thesis have been published in the papers listed in the publications section. This thesis has not been submitted, in whole or in part, for a degree at this or any other university.

Marcus Reginald Gallagher, June 30, 1999 (Revised January, 2000).

Abstract

The Multi-Layer Perceptron (MLP) is one of the most widely applied and researched Artificial Neural Network model. MLP networks are normally applied to performing supervised learning tasks, which involve iterative training methods to adjust the connection weights within the network. This is commonly formulated as a multivariate non-linear optimization problem over a very high-dimensional space of possible weight configurations.

Analogous to the field of mathematical optimization, training an MLP is often described as the search of an error surface for a weight vector which gives the smallest possible error value. Although this presents a useful notion of the training process, there are many problems associated with using the error surface to understand the behaviour of learning algorithms and the properties of MLP mappings themselves. Because of the high-dimensionality of the system, many existing methods of analysis are not well-suited to this problem. Visualizing and describing the error surface are also nontrivial and problematic. These problems are specific to complex systems such as neural networks, which contain large numbers of adjustable parameters, and the investigation of such systems in this way is largely a developing area of research.

In this thesis, the concept of the error surface is explored using three related methods. Firstly, Principal Component Analysis (PCA) is proposed as a method for visualizing the learning trajectory followed by an algorithm on the error surface. It is found that PCA provides an effective method for performing such a visualization, as well as providing an indication of the significance of individual weights to the training process. Secondly, sampling methods are used to explore the error surface and to measure certain properties of the error surface, providing the necessary data for an intuitive description of the error surface. A number of practical MLP error surfaces are found to contain a high degree of ultrametric structure, in common with other known configuration spaces of complex systems. Thirdly, a class of global optimization algorithms is also developed, which is focused on the construction and evolution of a model of the error surface (or search space) as an integral part of the optimization process. The relationships between this algorithm class, the Population-Based Incremental Learning algorithm, evolutionary algorithms and cooperative search are discussed.

The work provides important practical techniques for exploration of the error surfaces of MLP networks. These techniques can be used to examine the dynamics of different training algorithms, the complexity of MLP mappings and an intuitive description of the nature of the

error surface. The configuration spaces of other complex systems are also amenable to many of these techniques. Finally, the algorithmic framework provides a powerful paradigm for visualization of the optimization process and the development of parallel coupled optimization algorithms which apply knowledge of the error surface to solving the optimization problem.

Acknowledgments

Coming to the University of Queensland to pursue a PhD has been a richer experience than I could have hoped for. Firstly, I would like to thank my supervisor, Prof. Tom Downs, for giving me an opportunity to reach this goal, and for his direction, support and advice over the course of my candidature.

I have had the pleasure of working with many people in the Department of Computer Science and Electrical Engineering (and its former ancestor departments) during my candidature. In particular I would like to thank Len Allen, Dr. Hans Andersen, Kate Andersen, Ajantha Atourakale, Dr. Alan Blair, Chris Boucher, Nick Comino, Dr. Marcus Frean, Rachel Fredman, Jennifer Hallinan, Clary Harridge, Helen Lakadis, Dr. Steve Lawrence, Tracey Miller, Dr. Hugo Navone, Alex Pudmenzky, Dr. Guy Smith, Peter Stratton, Dr. P. Suganthan, Brad Tonkes, Dr. Brett Watson, Gerri Weier, A/Prof. Janet Wiles, Kath Williamson, Ian Wood, Dr. Steven Young and Jean Zhu.

The work in this thesis has also benefited from conversations with Shumeet Baluja, Rob Dunne, Karl Gustafson, Leonard Hamey, Wim Hordijk and Terry Jones.

This work has been supported by a Departmental Scholarship from the former Department of Electrical and Computer Engineering, and by a University of Queensland Postgraduate Research Scholarship.

Finally, I would like to thank my partner Angela; for putting up with my studies and never giving up on me, my parents, Reg and Gae Gallagher; my brother Dion; Pop, Nan, Grandpa and Grandma for their constant love, support and encouragement. This thesis is dedicated to my family.

List of Publications

- Marcus Gallagher, Marcus Frean and Tom Downs. Real-Valued Evolutionary Optimization using a Flexible Probability Density Estimator. In W. Banzhaf et al., editors, *Proc. Genetic and Evolutionary Computation Conference (GECCO'99)*, pp 840-846, 1999. Morgan Kaufmann Publishers, San Francisco, CA.

- Marcus Gallagher, Tom Downs and Ian Wood. Ultrametric Structure in Autoencoder Error Surfaces. In L. Niklasson et al., editors, *Proc. Eighth International Conference on Artificial Neural Networks (ICANN'98)*, pp 177-182. 1998. Springer, London.

- Marcus Gallagher and Tom Downs. On Ultrametricity in Feedforward Neural Network Error Surfaces. In T. Downs et al., editors, *Proc. Ninth Australian Conference on Neural Networks*, pp 236-240, 1998. University of Queensland.

- Marcus Gallagher and Tom Downs. Weight space learning trajectory visualization. In M. Dale et al., editors, *Proc. Eighth Australian Conference on Neural Networks*, pp 55-59, 1997. Telstra Research Laboratories.

- Marcus Gallagher and Tom Downs. Visualization of learning in neural networks using principal components analysis. In B. Verma and X. Yao, editors, *Proc. International Conference on Computational Intelligence and Multimedia Applications*, pp 327-331, 1997. Griffith University.

Contents

List of Figures

List of Tables

Abbreviations

ANN	Artificial Neural Network
AM	Apparent Minima
CDA	Canonical Discriminant Analysis
CG	Conjugate Gradient
EA	Evolutionary Algorithm
EDA	Exploratory Data Analysis
EM	Expectation-Maximization
EP	Evolutionary Programming
ES	Evolutionary Strategy
GA	Genetic Algorithm
IQR	Inter-Quartile Range
KLT	Karhunen-Loeve Transformation
LM	Levenberg-Marquardt
MLP	Multi-Layer Perceptron
MSE	Mean-Squared Error
NFL	No Free Lunch
PBIL	Population-Based Incremental Learning
PC	Principal Component
PCA	Principal Component Analysis
PV	Principal Value
QN	Quasi-Newton
SA	Simulated Annealing
SSE	Sum-of-Squares Error
SVD	Singular Value Decomposition
XOR	Exclusive OR

List of Symbols

D	Set of data samples
E	Error function
N	Number of weights
N_i	Number of inputs
N_h	Number of hidden units
N_o	Number of output units
T	Training Set
W	Set of weight vectors
$N_i - N_h - N_o$	MLP network with N_i inputs, N_h hidden units and N_o outputs
$\mathcal{N}$	Gaussian Distribution
$\mathcal{U}$	Uniform Distribution
a	Output activation of MLP unit
d	Distance measure
m	Number of components in a kernel probability density
t	Time index
q	Euclidean distance between two points
r	Correlation coefficient
s	Correlation coefficient (ultrametricity)
s_r	Rank Correlation coefficient (ultrametricity)
$\mathbf{w}$	Weight vector
$\mathbf{w}^*$	A global minimum weight vector for an MLP error surface
$\mathbf{x}$	Input Vector
$\mathbf{y}$	Network Output vector
z	Weighted sum of inputs to MLP unit
Σ	Covariance matrix
Ψ	MLP Network Mapping
Φ	Probability density mapping function
δ_j	BP error signal for MLP unit j
η	Learning Rate Parameter
γ	Sampling ratio - Points sampled:Points accepted

λ	Number of offspring in an ES
μ	Number of parents in an ES
π	Mixture model mixing coefficient
τ, τ'	ES variance perturbation constants

Chapter 1

Introduction

This chapter introduces the area of research considered in this thesis. The research issues considered and the methodology used to address these issues are then summarized. An outline of the following chapters is then given. The broad scope and limitations of the thesis are defined. Finally, the major original contributions of this dissertation are stated.

1.1 Overview

The field of Artificial Neural Networks (ANN's) is one aspect of research into the study of " ... *computation and cognitive modelling on neurally-inspired mechanisms* ... " [178] usually referred to under the broad heading of *Connectionism.* Connectionism has its origins in the 1940's, and has undergone a colourful history of development (see [151] and [178] for reviews). An extremely diverse group of researchers have an interest in some aspect of connectionism, including computer scientists and engineers, mathematicians and statisticians, physicists, cognitive scientists, neuroscientists, psychologists and linguists. The field as such is a highly cross-disciplinary one, bringing researchers from different backgrounds, with different motivations and different goals, to the idea of distributed, highly connected systems of simple processing units. Motivated originally by the structure of the mammalian brain, ANN models are many and varied [6].

Although the modern Von Neumann based computer architecture is capable of (increasingly) high speed computation, there remain many tasks which are very difficult for computers, but yet are accomplished easily by the human brain (e.g. complex motor skills, pattern recogni-

tion tasks). ANN models represent a bottom-up approach to the investigation of architecturally brain-like systems. By constructing simple models of neural architectures, it is hoped that an understanding can be gained as to how such an architecture is capable of such complex tasks. Even if the models are not biologically plausible, they hold the promise of increased computational capability and understanding of the interaction of complex systems in general.

This thesis is concerned with one of the most well-known ANN models: the Multi-Layer Perceptron (MLP). An MLP is a network of simple processing units arranged into a hierarchical model of layers. The units (neurons or nodes) in the first (input) layer are connected to nodes in the following layer(s) (hidden layers), to the final layer (the output layer). Numerical input vectors of (patterns) are presented at the input layer, and activity flows through the network to the output layer. Connections have a numerical weight value associated with them, and the signal transmitted via a connection is multiplied by the weight value. Each unit computes some function of the sum of its weighted inputs, and transmits the result through its output connections. In this way an MLP implements a mapping from input space to output space.

The backpropagation algorithm [203] provides a means of training the network to perform supervised learning tasks. Supervised learning starts with the presentation of a set of example input patterns to a learning system. The learners output is then compared with the known correct output for each pattern, and some adjustments are made so as to improve the response of the learner to those patterns. In an implementation, the goal is not simply to learn the patterns in the training set (which could be accomplished by a lookup table), but to learn in some sense the characteristics of the mapping, so as to be able to generalize (i.e. produce correct responses for patterns that the network has not been trained on).

Since the development of backpropagation in the mid-1980's, MLP's have been applied to a diverse range of practical problems through supervised learning, such as adaptive filtering and signal processing, forecasting, classification, speaker identification and speech recognition, handwriting and character recognition, control and telecommunications problems (see, e.g. [6]). MLP's have been very successful in these applications, producing results which are at least competitive and often surpassing other existing computational learning models.

1.2 Research Problem, Methodology and Justification

The aim of this dissertation is to develop practical analysis techniques for a specific problem, which remain general enough to be applicable to a broad existing class of problems. The specific problem is the training of a MLP network. This corresponds to a high-dimensional, non-linear optimization problem over the space of network weight parameters. Three main kinds of techniques are investigated: scientific visualization methods, statistical sampling methods and probabilistic-modelling optimization algorithms.

Major research efforts into MLP's have concentrated on the design of improved training algorithms and techniques to improve the performance of MLP's. Literally hundreds of algorithms have been proposed which claim to offer such improvements. Unfortunately, a number of difficulties have also become clear:

- The success of any MLP training algorithm is dependent on the supervised learning problem (i.e. the dataset).
- The success of any MLP training algorithm is also dependent on many experimental factors, such as user-adjustable parameters, the topology of the network and preprocessing of the training data.
- Empirical comparisons of algorithms have often been less than satisfactory, as have the original empirical testing of the algorithms accompanying their proposal.

It is important therefore to explore techniques that can be used to analyze the dynamics of training algorithms, compare the operation of training algorithms and to better understand the relationship between the dataset, network model, prior knowledge and algorithm in the MLP training problem.

Scientific visualization broad field of research, which has received little application in the domain of neural networks. Visualization has proven to be a powerful tool for exploratory data analysis in other fields such as fluid dynamics, yet MLP training presents difficulties of its own for visualization, in particular for data of very high dimensionality. A number of visualization techniques are developed in this thesis which address this issue.

The MLP training problem is a particular kind of optimization problem. Researchers have gradually become aware of the importance of incorporating any available prior knowledge of

the problem domain into the solution process, to obtain improved results. Nevertheless, it is often a difficult task to understand the implicit assumptions and conditions which often exist in the realization of systems such as MLP's. Statistical sampling methods provide techniques for investigating the nature of the given problem which are generally applicable and which scale up to systems of practical sizes.

Typically, training algorithms make many assumptions, and do not deal directly with the properties of the solution space. From the viewpoint of this dissertation, algorithms which model the structure of the search space offer a number of benefits. An algorithmic framework is presented in this thesis which allows the expression of a variety of existing algorithms in this context, and facilitates the development of new algorithms which utilize probabilistic modelling techniques.

It has been realized that the above issues which occur in MLP training appear in other fields of research involving complex systems of large numbers of adjustable parameters. It is therefore desirable to develop techniques which remain applicable to other systems of this kind.

1.3 Outline

An outline of the remainder of this thesis is as follows. Chapter 2 introduces the MLP network architecture, with particular consideration given to the MLP training task and its framing as an optimization problem. The nature of optimization problems in general is then discussed, as is the work on algorithms and heuristics for improving MLP training performance. Chapter 3 is concerned with scientific visualization, in particular its application to high-dimensional, multivariate systems and MLP training tasks and algorithms. The technique of Principal Component Analysis is applied to learning trajectory data and is found to be an effective method with good scaling properties to large systems. In Chapter 4, the error surface concept is discussed in detail, comparing and contrasting existing results to provide a comprehensive description of the known properties of the MLP error surface. Statistical sampling methods are employed to further investigate error surfaces. The results are compared with those from related problems in other fields. In the final section of Chapter 4, a hierarchical, third-order statistical property known as ultrametricity is shown to exist in MLP error surface data samples. Chapter 5 develops an algorithmic framework for the specification of a new class of global stochastic optimization al-

gorithms. Such algorithms involve the explicit creation and adaptation of a probabilistic model of the search space (and hence the error surface), and have a number of attractive properties. Chapter 6 concludes and provides direction for future work.

1.4 Scope and Limitations

It is important to note that this thesis does not itself attempt to provide new training algorithms which (claim to) provide superior MLP training performance. In fact, the work seeks to highlight and avoid the problems associated with such an approach, instead looking toward the goal of gaining a deeper understanding of the nature of the problem (MLP training), solution space (error surface) and the factors that they are dependent on (network topology, training set, error function, prior knowledge, heuristics).

As a consequence, many issues concerning generalization, training data preprocessing and feature extraction are beyond the scope of this thesis. Although the focus in this work is the MLP, it is clearly important to develop techniques that are applicable to other models or systems with the same general characteristics.

1.5 Original Contributions

The major contributions of this thesis are as follows:

1. A comprehensive review of the application of multivariate scientific visualization to artificial neural networks, in particular the MLP.

2. An original approach to the visualization of the trajectories and temporal behaviour of MLP training algorithms on the error surface, using Principal Component Analysis.

3. A summary and analysis of the existing results concerning the error surface of MLP networks.

4. A unified presentation of the ideas scattered throughout a wide range of literature on the analysis of the structure of the search spaces of optimization problems and other complex systems.

5. Application of statistical sampling methods that occur unsystematically in the literature of various fields to MLP error surfaces.

6. An original adaptation and application of the Fitness-Distance Correlation method for statistical analysis and visualization of the search space, to continuous MLP error surfaces.

7. The detection of ultrametric structure in samples of low-lying regions and "apparent minima" of a number of MLP error surfaces, through statistical sampling techniques. This result provides a new connection between the continuous search space of the MLP error surface and the discrete search spaces that exist in combinatorial optimization problems, evolutionary biology and statistical physics.

8. A new algorithmic framework which is adequate for the description of a large class of optimization algorithms that view the construction of a probabilistic model of the search space as an integral part of the optimization task. It is also shown using several examples how a number of existing algorithms can be formulated in this manner.

9. Two new optimization algorithms for the modelling of the error surface that use more powerful probability density estimation techniques than have previously been employed. These algorithms provide an implementation of parallel, coupled stochastic search for continuous optimization problems.

Chapter 2

Multi-Layer Perceptrons, Training Algorithms and Optimization

This chapter introduces the area of artificial neural networks, in particular the Multi-Layer Perceptron (MLP) network model which has developed as an effective and powerful model for performing supervised learning tasks. It is then shown that training an MLP can be expressed as a non-linear optimization problem over the space of adjustable network weight parameters. The remainder of the thesis is concerned with issues based on the notion of the error surface which follows from this problem formulation.

The area of mathematical optimization is discussed in light of the given optimization problem, with a focus on issues of terminology and various concepts relevant to the MLP error surface and training. The backpropagation algorithm is viewed as a means of implementing the gradient descent optimization algorithm, through calculation of the derivative of the error function. A number of other algorithms developed in optimization research are discussed which have been applied to training MLP's. The main problems particular to training an MLP are also introduced. These problems are discussed in the context of the No Free Lunch Theorems, the nature of global and local optimization problems and the issues of assumptions and prior knowledge.

2.1 Overview

The neurons which are found in the human brain are many and varied, and much is yet to be understood about them [5]. Artificial neurons are simplified models based on abstractions of the known properties of biological neurons. While the biological plausibility of artificial neural network (ANN) models is rather doubtful, research has shown that ANN's are complex, non-linear systems which display many interesting and useful properties.

Neuron-like elements have also been developed as computational and signal processing devices in the fields of computer science and electrical engineering. These devices include the binary threshold elements (also called linear threshold units), layered networks of binary threshold elements [162] and the adaptive linear combiner [252].

2.2 Artificial Neurons

The work of Warren McCulloch and Walter Pitts in 1943 is seen as something of a starting point for research into ANN's. In an effort to understand the workings of biological neural systems, McCulloch and Pitts examined networks of simplified *binary threshold elements*. Their artificial neurons or *units* are based on the all-or-nothing property of neuron firing, in a discrete time scale (see [5]). A diagram of a McCullogh-Pitts neuron is shown in Figure 2.1. Each unit has N inputs x_i, each of which has an associated *weight* value w_i. This artificial neuron computes a function of the weighted sum of its inputs and outputs a '1' if this sum exceeds some given threshold value θ, and a '0' otherwise

$$\Theta(a) = \begin{cases} 1 & \text{if } a - \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

where

$$a = \sum_{i=1}^{N} x_i w_i$$

is the weighted sum of inputs. Θ is known as the Heaviside step function.

McCullogh and Pitts were able to show that networks of such units could be constructed to implement the Boolean AND, OR and NOT operations, and hence any Boolean function.

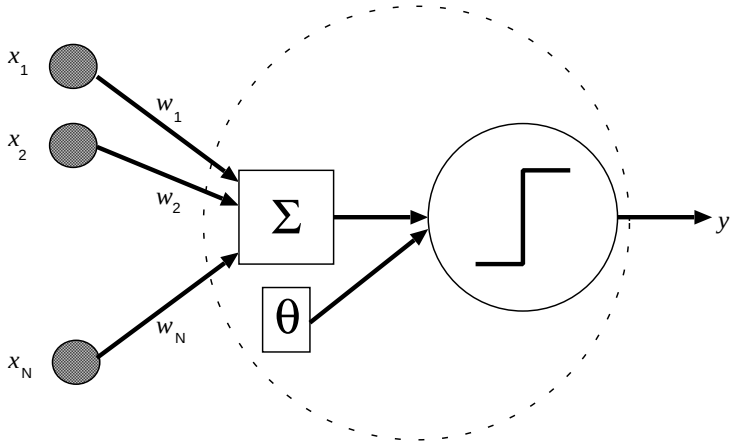

Figure 2.1: McCullogh-Pitts neuron.

In general, an artificial neuron unit computes a (possibly non-linear) function of a weighted sum of inputs

$$y = f\left(\sum_{i=1}^{N} x_i w_i - \theta\right).$$

There are a number of variations on the threshold-like *activation function* used in ANN's[1]. A symmetric output is provided by replacing the Heaviside function Θ with the $sgn()$ (also called signum) step function

$$sgn(a) = \begin{cases} 1 & \text{if } a - \theta > 0 \\ -1 & \text{otherwise} \end{cases}$$

These units are sometimes called Ising perceptrons, in an analogy to magnetic spin glass systems of statistical physics [157]. Another important class of activation function are the *sigmoidal* functions, such as the logistic

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

and the hyperbolic tangent

$$f(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}.$$

[1]A number of other classes of activation functions have been used in the literature, such as Radial Basis Function neurons (see, e.g. [32]). These functions are not considered in this thesis.

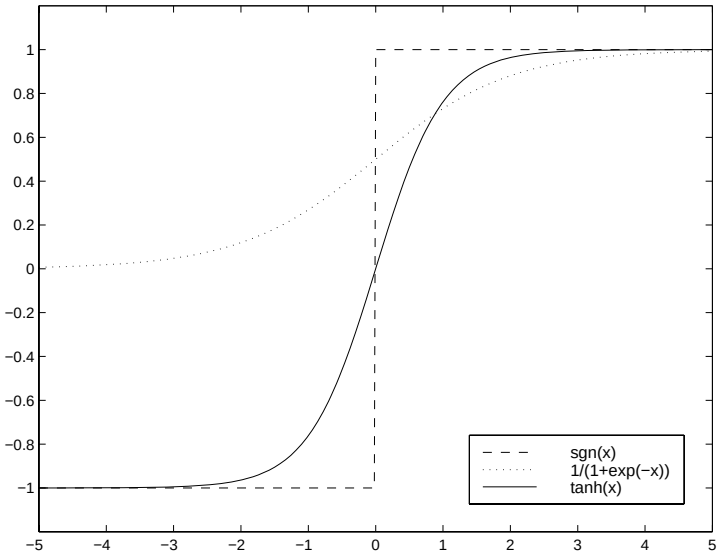

Figure 2.2: Commonly used artificial neuron activation functions.

These functions provide a smooth, continuous version of a threshold function, which is differentiable. These properties are advantageous for a number of reasons, as discussed in later sections. Figure 2.2 shows the signum, logistic and hyperbolic tangent functions.

Sigmoidal units are also able to incorporate a threshold or bias value, that effectively translates the center of the sigmoid to some arbitrary value. This bias is conveniently implemented using an additional input x_{N+1} whose value is permanently clamped to ± 1. The value of the associated weight w_{N+1} then determines this offset, which forms part of the weighted sum of inputs to the unit.

2.3 The Multi Layer Perceptron Architecture

Consider a single layer of N_o artificial neuron units, with each unit fully connected to each of the N_i common inputs (plus the bias input unit). This network provides mappings of the input patterns to multi-dimensional output vectors $\mathbf{y} = (y_1, \ldots, y_Q)$, and is referred to as a *single-layer perceptron network*. The Multi-Layer Perceptron (MLP) network allows the additional extension to multiple layers of units, with the outputs of the first layer of units becoming the inputs of the next, and so on. Whatever the structure of the network, information flows in one direction, from the network inputs, forward towards the final (output) layer of units, and produces the network output associated with that input.

Figure 2.3 shows the general architecture of the MLP. The inputs of the network are referred

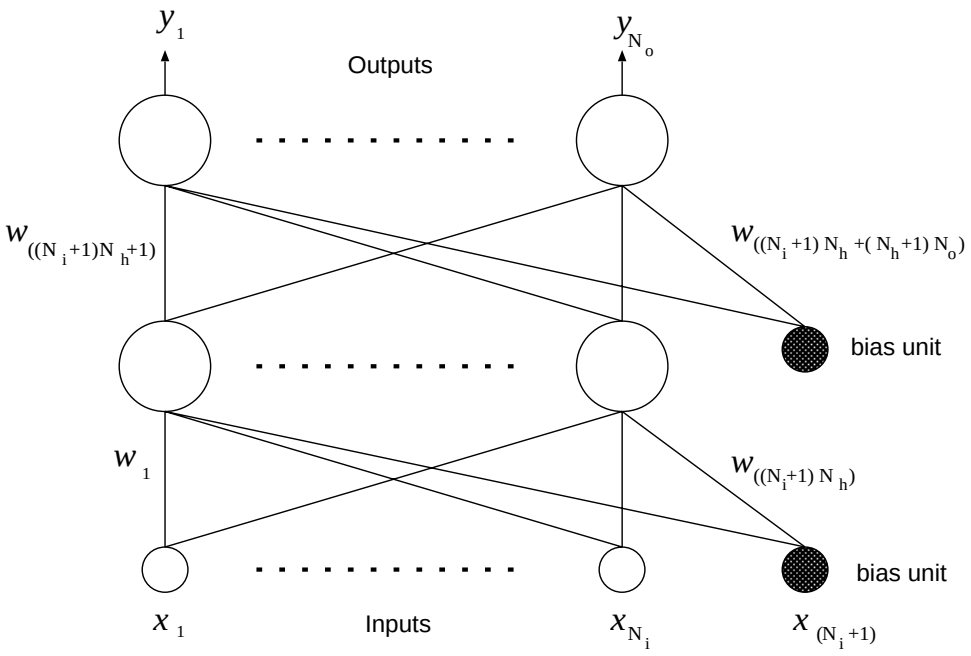

Figure 2.3: Multi-layer Perceptron General Architecture.

to as the input layer - these inputs are simply points at which inputs are "applied" to the network. Signals propagate forward from the input layer, through one or more intermediate or hidden layers of units, to the final, or output layer of nodes. The outputs of these nodes are then taken as the output of the network. Connections are unidirectional - there is no feedback or cycles in the network. An MLP is said to be *fully connected* if every node in a given layer is connected to every node in the following layer. Skip connections, which allow direct weights between nodes in non-adjacent layers (e.g. from the inputs to the outputs in Figure 2.3), are also sometimes allowed. In Figure 2.3 the network has N_i inputs, a single hidden layer of N_h units and N_o output units. The network is fully-connected, with no skip connections. In this thesis, the notation N_i-N_h-N_o is used to denote the configuration of a MLP with a single hidden layer (e.g. a 2-2-1 MLP has 2 inputs, 2 hidden units and a single output). MLP's with multiple hidden layers are not used.

MLP network weights are normally taken to be real-valued. Inputs can also be real, but will depend on other factors such as the nature of the data, preprocessing and feature extraction methods. Sigmoidal output unit activations can be used to approximate binary (e.g. $\{0, 1\}, \{-1, +1\}$) values. It is possible to use a linear activation function (i.e. the output is the weighted sum) for output units when the target outputs are real-valued. In this thesis, the MLP networks used have sigmoidal activation functions in the hidden and output layers, unless

otherwise stated.

The network mapping Ψ can be written in vector notation as

$$\mathbf{y} = \Psi(\mathbf{x}, \mathbf{w}) = f_o(f_h(\mathbf{x} \cdot \mathbf{w}_{ih})) \cdot \mathbf{w}_{ho})$$

where $\mathbf{x}$ is the input, $\mathbf{y}$ is the output of the network, and $f_o(\cdot), f_h(\cdot)$ are the activation functions of the output and hidden units respectively. Denote the vector of all weights in the network as

$$\mathbf{w} = (\mathbf{w}_{ih}, \mathbf{w}_{ho})$$

This "global" weight vector can be decomposed into a vector of the first layer of weights

$$\mathbf{w}_{ih} = (\mathbf{w}_1, \ldots, \mathbf{w}_{((N_i+1)N_h)})$$

and a vector of the second layer of weights

$$\mathbf{w}_{ho} = (\mathbf{w}_{((N_i+1)N_h+1)}, \ldots, \mathbf{w}_{((N_i+1)N_h+(N_h+1)N_o)})$$

including N_h weights from a bias input unit, and N_o weights from a bias hidden unit.

2.4 Computation using MLP's

2.4.1 Supervised Learning

It is clear that an ANN performs some mapping from its (vector) input space to its (vector) output space, through a (non-linear) transformation given by the weights and the activation functions of the network. The network weights are the adjustable parameters of the system - changing the values of the weights in the network changes the mapping itself.

This thesis is concerned with using MLP's to perform *supervised learning* tasks. Here a set of k input vectors together with the correct output pairs is given

$$T = \{(\mathbf{x}^1, \mathbf{t}^1), \ldots, (\mathbf{x}^k, \mathbf{t}^k)\}.$$

These data pairs represent examples of a mapping which it is desired to implement using

an MLP. For a given input $\mathbf{x}^j$, the network produces an output vector $\mathbf{y}^j = \Psi^j(T^j, \mathbf{w})$. The supervised learning task is to adjust the weights of the network such that for each input vector of this training set which is presented to the network, its output approximates the desired output vector as closely as possible. In order to do this, a cost or error function is defined, such as Mean Squared Error (MSE)

$$E^k = \frac{1}{2}\|\mathbf{y}^j - \Psi^j(T^j, \mathbf{w}))\|^2$$

which can be summed over the entire training set

$$E = \frac{1}{k}\sum_{i=1}^{k} E_i.$$

The Sum-of-Squares Error (SSE) is the unnormalized version, $E_{SSE} = \sum_{i=1}^{k} E^i$.

While MSE is the most commonly used error function in MLP applications, there are many other choices which are well-founded, depending on the nature of the training data and the desired output of the network (e.g. [32], Chap. 6). The MSE and SSE error functions are used in this thesis, although the techniques and results developed are not dependent on this choice and generalize trivially to alternative error functions.

2.4.2 Generalization and Model Complexity

Training an MLP to learn a good representation of the training set is the direct action of supervised learning. However this is not the true goal of learning, since if this were the case the problem could be solved much more efficiently with a lookup table. In practical applications, it is expected that the network is, after training, able to generate correct outputs for input data patterns that were *not* part of the training set. In other words,

> *. . . the goal of supervised learning is not to learn an exact representation of the training data itself, but rather to build a statistical model of the process which generates the data.* (Bishop [32], p. 332)

While a lookup table performs perfectly on the training set, it has no generalization capability. MLP's have shown impressive generalization performance in a wide range of applications, which is no doubt the main reason for the amount of attention they have received.

A significant amount of research into supervised learning continues to be devoted to understanding and improving the generalization performance of learning models. It is well known that a trade-off exists between the complexity of the learning model and its ability to generalize - the *bias-variance* tradeoff [78]. Generally speaking, this formulation shows that the effective complexity of the model should be well-matched to the data itself. Too simple a model can result in a poor approximation of the data, as the model cannot capture all of the information in the data. However, too complex a model may result in poor generalization performance as the model is flexible enough to fit any anomalies, noise, etc. in the training data, which are not representative of the underlying data source.

Much research is concerned with minimizing bias and variance in a supervised learning model. Statistical research includes a number of areas that can be used to improve generalization of any kind of supervised learner (not just an MLP), such as regularization, early stopping techniques, smoothing and training with noise, and combining learning models using ensemble and committee techniques (see, e.g. [91]). In the context of MLP research, improved learning methods such as *weight decay* can be seen as the implementation of a statistical approach [32]. Modifications to the structure of the network itself via constraints (e.g. weight sharing) or during training (growing or pruning algorithms) also attempt to address the issue of model complexity.

In this thesis the problem of generalization using MLP's is not considered further. Although generalization is of major importance, it is not required to study issues concerning the error surface dependent on some given training set. However, it should be noted that the techniques developed here may be useful in studying aspects of the generalization problem.

2.4.3 Representational Capabilities of MLP's

An important consideration is the kinds of input-output mappings or functions that can be represented with an MLP. It has been shown that MLP's are *universal approximators*, that is, they capable of approximating essentially any continuous function to an arbitrary degree of accuracy (see, e.g. [91] for a summary and references). This result means that, given a single hidden layer of units, weight values exist such that any given training set can be approximated arbitrarily well by the network. Unfortunately, this result does not provide a means of finding the particular weight values, or the number of hidden units required to achieve this approximation. Also, the number of units in the hidden layer is required to grow arbitrarily large for some

functions [32].

It might be perceived that there is no need to consider MLP's with more than one hidden layer, given the universal approximator result. However, it is known that arranging a given number of units into multiple layers allows an MLP to implement more complex mappings [79], and that a network with two hidden layers typically requires less units (and hence weights) to implement a given function to any degree of accuracy [230]. Training problems such as the two-spirals dataset [136] exist where more than one hidden layer leads to improved results, and others can be easily constructed [44, 51].

One further point regarding the universal approximation result is that in practice, as discussed in the previous section, generalization is the desired outcome of training. Approximation of the training set to great accuracy may not necessarily produce a network with good generalization performance.

2.5 MLP Training as an Optimization problem

As shown in the previous section, learning in MLP's involves iteratively adjusting the weights in the network so as to approximate an unknown mapping, given a set of example input-output patterns representing the mapping. The network learns by experience to improve its performance on the given task. This task is conveniently and usually formulated [24, 32] as an *optimization problem*, as follows. The search space W is the set of all feasible weight configurations (vectors) of a network. The error function E assigns a real number to each feasible point $\mathbf{w} \in I\!R^N$

$$E : W \rightarrow I\!R.$$

E is measured on the training set and is a measure of the network performance at producing the output values in the training set in response to the training set input values. The optimization problem is then to find a vector $\mathbf{w}^*$ which minimizes the value of the error function

$$\mathbf{w}^* = arg \min_{\mathbf{w}} E(\mathbf{w}). \tag{2.1}$$

2.5.1 The Error Surface Metaphor

The optimization task can be thought of as a search of the space of all possible configurations of the adjustable parameters of the problem (in this case the weights of an MLP) for a point which best satisfies the optimization condition (Equation 2.1). In the neural networks literature, the search space is referred to as *weight space*. The value of the error function at each point in weight space can also be considered as the height of a multidimensional surface sitting above weight space. This surface is known as the *error surface* and is the main focus of this thesis.

Research involving optimization problems is large and diverse, as is the terminology used to describe the "weight space" and "error surface" concepts. In the optimization literature, the error function is known as the cost function, and the error surface is the cost surface or landscape. The term *objective function* is also used. Computer scientists tend to refer to the parameter (in this case, weight) space as the solution or search space. Physicists prefer the configuration space, and energy landscape, since concern there is often with minimizing the energy of a physical system. Researchers in Evolutionary Computation and Biology refer to evolution as adapting a species towards higher fitness - the (inverse) maximization problem. This term leads to the notion of the fitness landscape and fitness function.

Because the terminology is so diverse, this thesis does not attempt to reconcile or replace these terms with a common standard. Each terminology is well-suited to the context to which is applies. Hence, much of this terminology will be used depending on the subject discussed. It should be clear however that the error function, error surface and weight space concepts in MLP research are simply terms for generic concepts of the optimization problem, which is much more general. Further discussion is deferred until Chapter 4.

In Figure 2.4 a diagram is shown indicating the relationships between the training set of the supervised learning problem, the error function (e.g. MSE over the training set), the MLP (model used to perform the task) and the error surface. In a general mathematical optimization problem the picture is simpler; the error function and a number of adjustable parameters (taking the place of the MLP) are present, but there is no training set. The error surface is indirectly dependent on the training set, through the error function itself. It is important to consider that making changes to the training set, error function or MLP changes the structure of the error surface. Unfortunately, these changes cannot be easily quantified, because of the nonlinearity of the system and the dependence of the error surface structure on multiple factors. In general,

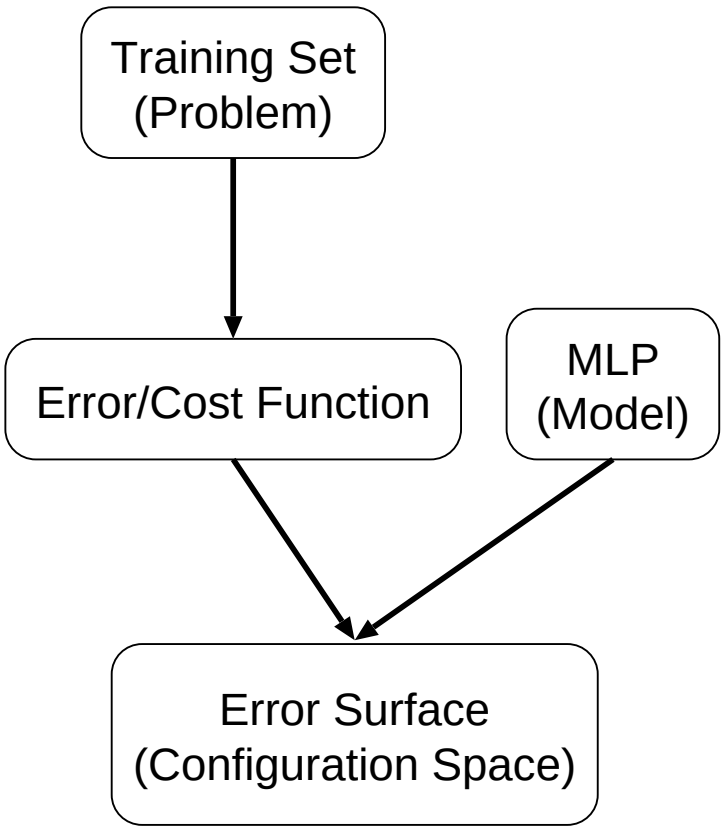

Figure 2.4: The relationship between the error surface, the MLP and the supervised learning problem.

Characteristic	Property	Classification
Number of control variables	One	Univariate
	More than one	Multivariate
Type of control variables	Continuous real numbers	Continuous
	Integers	Integer or discrete
	Both continuous real numbers and integers	Mixed integer
Problem functions	Linear functions of the control variables	Linear
	Quadratic function of the control variables	Quadratic
	Other nonlinear functions of the control variables	Nonlinear
Problem formulation	Subject to constraints	Constrained
	Not subject to constraints	Unconstrained

Table 2.1: A summary of classes of optimization problems.

increasing the size of the MLP model (e.g. the number of hidden units) leads to (at least, potential) increase in the model's representational capacity. Similarly, increasing the size of the training set can lead to a more complex problem. Unfortunately, these changes cannot be easily quantified, because of the nonlinearity of the system and the dependence of the error surface structure on multiple factors.

2.6 Optimization

The optimization problem stated above (2.1) is a particular kind of problem in the field of mathematical optimization or operations research. In the context of this field, training an MLP is a multivariate, continuous, nonlinear, unconstrained optimization problem. Many standard textbooks cover such optimization techniques, including [75, 189, 237, 253]. One possible categorization of optimization problems is shown in Table 2.1.

A number of useful definitions can be made with regard to such an optimization problem. A point which satisfies the optimization problem (2.1) is called a *global minimum*, where

$$E(\mathbf{w}^*) \leq E(\mathbf{w}) \; \forall \; \mathbf{w} \in W, \; \mathbf{w} \neq \mathbf{w}^*$$

It is also important to consider points which may not be minima of the entire search space, but which are so in some given region of interest in the space. These points are called *local minima*.

A local minimum is a point for which

$$E(\mathbf{w}^l) < E(\mathbf{w}) \; \forall \; \mathbf{w} \in N(\mathbf{w}^l, \epsilon), \; \mathbf{w} \neq \mathbf{w}^l$$

where $N(\cdot, \cdot)$ is the set of points within some small distance ϵ of $\mathbf{y}^l$, that is, the *neighbourhood* of $\mathbf{y}^l$. This minimum is a *strong* local minimum, because of the strictness of the inequality. A *weak* local minimum is similarly defined by relaxing the above, i,e

$$E(\mathbf{w}^l) \leq E(\mathbf{w}) \; \forall \; \mathbf{w} \in N(\mathbf{w}^l, \epsilon), \; \mathbf{w} \neq \mathbf{w}^l.$$

Weak minima imply a region (which may be a line or a higher-dimensional area) of the search space which is a local minimum[2].

If the objective function is smooth and has first derivatives, critical or stationary points of the function can then be defined as those for which

$$E'(\mathbf{w}) = 0$$

where $E'(\mathbf{w}^*)$ is a vector with elements of the form

$$E'_i(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial w_i}, \; 1 \leq i \leq n.$$

If the the objective function is twice differentiable, the *Hessian* matrix $E''(\mathbf{w})$ of partial second derivatives, containing elements

$$E''_{ij}(\mathbf{w}) = \frac{\partial^2 E(\mathbf{w})}{\partial w_i \partial w_j}, 1 \leq i, j \leq n$$

can be used to classify the stationary points of the function. The Hessian is also symmetric in this case. If $\mathbf{w}^l$ is a strong local minimum of $E(\mathbf{x})$ then the Hessian matrix is positive-definite (i.e. all the eigenvalues of $E''(\mathbf{w}^*)$ are strictly positive). Similarly, if all the eigenvalues of $E''(\mathbf{w}^*)$ are strictly negative at $\mathbf{w}^*$, then $\mathbf{w}^*$ is a strong local maximum. A Point where the Hessian is positive semi-definite (negative semi-definite) (i.e. $E''(\cdot)$ has all non-negative (non-positive) eigenvalues) may be a weak minimum (maximum). However, these points may also

[2]In a practical situation, finite precision for floating-point numbers may induce such weak minima.

be weak saddle points. Finally, if the Hessian is indefinite (i.e. has both positive and negative eigenvalues), the point is definitely a saddle point.

In the neighbourhood of a minimum (or maximum), the error surface will be locally convex. Local convexity corresponds to a Hessian which is positive definite at the minimum. In most practical situations, it is not known if a given minimum is in fact the global minimum. Verification of this property requires an exhaustive search of the error surface. MLP error functions are defined to be non-negative

$$E(\mathbf{w}) \geq 0 \; \forall \; \mathbf{w} \in W$$

which places a lower bound on the error value at a global minimum.

If the error function corresponds to a closed-form mathematical expression, then it is often possible to solve the function analytically to determine the location and error value of the critical points. However in the case of a parameterized system such as an MLP, no closed-form expression exists, and the first and second derivatives of the error function must be calculated for each given point on the error surface. It follows that critical points of the error surface may only be approximately determined by numerical methods, that is to some given accuracy. To ease the discussion of the features of error surfaces that are developed from human experience of the geometry of three-dimensional surfaces and the features produced by critical points, the following definitions are introduced (based on those of [67] and [89]):

Definition 2.1 *A non-ascending trajectory is a continuous function from a point* $\mathbf{w_i}$ *to* $\mathbf{w_j}$ *on the error surface, such that the error function* E *is everywhere non-increasing along the trajectory.*

Definition 2.2 *A gradient-descent trajectory is a non-ascending trajectory which is obtained by moving in the direction of negative gradient with an arbitrarily small step-size* ξ*.*

Definition 2.3 *A region of attraction of a local minimum is the set of points from which a non-ascending trajectory exists to that local minimum.*

Note that this property does not guarantee that such a gradient descent method will reach the local minimum - saddle points may exist in regions of attraction which trap gradient-descending trajectories.

Definition 2.4 *A basin of attraction of a local minimum is the set of points from which the gradient-descending trajectory is guaranteed to converge to the local minimum.*

2.6.1 Local and Global Optimization Methods

The optimization problem stated in Equation 2.1 above is referred to as the *global* optimization problem. Solution of the global optimization problem requires the location of a point $\mathbf{w}^*$ at which the cost function attains its smallest value. This problem is in general intractable and ill-posed [237]. In general there is no way to determine that the global minimum has been reached. In contrast, methods which locate a *local* optimum of the cost function attain this goal at points where the gradient equals zero[3] (as stated above). The convergence of a local method is therefore decidable by showing that the distance between trial solutions and a local minimum approaches zero as the number of trials grows. A simpler requirement of probabilistic convergence for global optimization methods is the requirement that in the limit of execution time, the probability that the global optimum is attained approaches 1 [219].

From a practical viewpoint however, the optimization problem can be reformulated in the following way:

> *There exists a goal (e.g. to find as small a value of $E(\cdot)$ as possible), there exist resources (e.g. some number of trials), and the problem is how to use these resources in an optimal way.* (Törn and Žilinskas [237], p. 7)

It is thus convenient to consider local methods as those which are trapped[4] by local minima, and global methods as those which are able to escape[5] local minima.

The literature on both local and global optimization techniques is extensive [189, 210]. In the sections below a number of these techniques are mentioned in the context of their application to the MLP training problem.

[3] If the cost function is differentiable.

[4] Meaning that given enough time, the method reaches a local minimum, becomes arbitrarily close, or remains in the region of attraction of the minimum.

[5] Meaning that given enough time, the method leaves the region of attraction of a local minimum.

2.6.2 Backpropagation

The first widely used method for training an MLP to perform supervised learning is known as the backpropagation algorithm [203]. Backpropagation was responsible for creating much interest in feedforward ANN's, and has facilitated the application of the MLP to an enormous range and number of supervised learning tasks.

Backpropagation provides a means of adjusting the weights in an MLP, given a set of training data. It allows the gradient of the error function to be calculated (composed of partial gradient information with respect to different groups of weights in the network), and implements gradient descent on the error function in weight space

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t + \Delta\mathbf{w}^t \\ \Delta\mathbf{w}^t &= -\eta E'(\mathbf{w}),\ \eta > 0 \end{aligned}$$

where η is a learning rate which can be chosen as a constant value.

Briefly, backpropagation is implemented as follows. To calculate the derivative of the error value E^k of pattern k, the input $\mathbf{x}_j$ is applied to the network, and the output activation is calculated for each unit using (forward propagation)

$$a_j = \sum_i w_{ji} z_i \tag{2.2}$$

$$z_j = f(a_j) \tag{2.3}$$

where z_i is the ith input of the unit, w_{ji} is the weight from unit i to unit j, a_j is the weighted sum of the inputs and z_j is the output activation of the unit activation function $f(\cdot)$. In the case of a hidden layer unit, the inputs z_i are the actual input patterns from the training set, while for the output layer units, the outputs z_j are the final outputs of the network.

The derivative with respect to w_{ji} is written as

$$\frac{\partial E^k}{\partial w_{ji}} = \frac{\partial E^k}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \tag{2.4}$$

since the error depends on w_{ji} only through the weighted sum a_j. Let the error signal (or "delta"

value) for the unit be

$$\delta_j \equiv \frac{\partial E^k}{\partial a_j} \tag{2.5}$$

Using (2.3)

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{2.6}$$

hence (2.4) simplifies to

$$\frac{\partial E^k}{\partial w_{ji}} = \delta_j z_i. \tag{2.7}$$

For the output units, the delta values can be written as

$$\delta_j \equiv \frac{\partial E^k}{\partial a_j} = f'(a_j)\frac{\partial E^k}{\partial y_j} \tag{2.8}$$

using (2.3) with $z_j = y_j$. For the hidden units, the chain rule can again be used, to write the delta values as

$$\delta_i \equiv \frac{\partial E^k}{\partial a_i} = \sum_{j=1}^{N_o} \frac{\partial E^k}{\partial a_j}\frac{\partial a_j}{\partial a_i}. \tag{2.9}$$

Using (2.5), (2.3) and (2.3), this simplifies to

$$\delta_i = f'(a_i)\sum_{j=1}^{N_o} w_{ji}\delta_j \tag{2.10}$$

The gradient of the error function for a training set is given by summing the gradients over all patterns in the set

$$\frac{\partial E}{\partial w_{ji}} = \sum_k \frac{\partial E^k}{\partial w_{ji}} \tag{2.11}$$

$$E'(\mathbf{w}^t) = \sum_k E'^k(\mathbf{w}^t). \tag{2.12}$$

Performing weight updates after the calculation of the gradient over the entire training set is referred to as *batch* backpropagation. A popular alternative is to update based on the gradient calculation with respect to a single training pattern

$$\Delta \mathbf{w}^t = -\eta E'_k(\mathbf{w})$$

which is known as *stochastic* or *on-line* backpropagation.

Although backpropagation implements simple gradient descent on the weight space of an MLP, the method has additional advantages [96] in that

- updates can be calculated using only local information (which may be important from the view of biological plausibility, hardware implementations or simulation on parallel hardware)
- the computational complexity of calculating the derivatives is $O(N)$ where N is the number of weights.

Perhaps the most important reason for the success of backpropagation is simply that it has worked reasonably well in a large number of artificial and real-world applications [138]. The reasons for this success are discussed in Section 2.7.4 below.

2.7 Other Algorithms for Training MLP's

Since the popularization of the backpropagation algorithm, a very large amount of research has been devoted to the development of algorithms that improve on backpropagation. The improvements which are most commonly sought are:

1. *Training speed* - algorithms which aim to attain some error value in as little time as possible
2. *Error value* - algorithms which aim to train the network to lower error values
3. *Generalization* - algorithms which aim to improve the generalization ability of the trained network
4. *Network Evolution* - algorithms which attempt to evolve the topology and/or connectivity of the network as part of the training process

Many algorithms attempt to directly address a number of these goals simultaneously. Also, it is not usually desirable to trade off these goals to a great extent, for example to accept a large decrease in generalization performance to gain an improvement in training time. In practice, it is normally desirable to maximize each of these goals as much as possible.

With regard to the training optimization problem, only the first two goals are relevant. Network evolving algorithms (see e.g. [42, 243, 159, 184, 244]; for surveys see [190, 259]) and methods which are predominantly focused on improving generalization performance (e.g. regularization, early stopping and training with noise [32, 167]) are beyond the scope of this thesis.

In this section, research into developing more effective training algorithms for MLP's is briefly reviewed. Some of the successes and problems of this area of research are also discussed.

2.7.1 Methods Based on Local Optimization

The majority of research into MLP training algorithms has considered methods based on local optimization techniques. The primary reason for this stems from the development of backpropagation, together with the relative computational cheapness of gradient information [32]. Gradient-based training algorithms have proven widely successful in a broad range of applications.

In addition to the on-line version of backpropagation, Rumelhart et al. introduce a *momentum* term into the backprop weight update equation [203], as one of the first and most well-known modifications to the algorithms. Many other algorithms have attempted to improve the convergence of the backprop update rule, for example through the adaption of the learning rate parameter η (*bold driver* [25, 242], *delta-bar-delta* [113]). Learning rate schedules have been developed from stochastic approximation methods to improve convergence to local minima [58]. Other well-known algorithms include *Quickprop* [74], *RProp* [193] and *SuperSAB* [218, 236].

A number of methods from the field of optimization have been adapted to the MLP training problem. These include Conjugate Gradient (CG) and Scaled Conjugate Gradient Methods, as well as methods based on second derivative information, for example the quasi-Newton (QN) and Levenberg-Marquardt (LM) algorithms [22, 26, 86, 108, 153, 217].

Good descriptions of many algorithms including the above can be found in a number of texts (e.g. [32, 91, 96, 194]). The sheer number of algorithms described in the literature makes

the task of a comprehensive review formidable, and is perhaps excessive for practical purposes, given the ad hoc nature of some algorithms. To the author's knowledge, Darr provides the only attempt at a study of this kind [59].

2.7.2 Methods Based on Global Optimization

Global optimization algorithms have also been proposed for training MLP's, the main motivation being to alleviate the problem of local minima on the error surface. This issue is discussed further in Chapter 4. In effect, this goal corresponds to the allowing increases of error value by the training algorithm.

Baba has adapted random search techniques from the field of global optimization [8]. Subsequently, Baba et al. [9] developed a hybrid global algorithm, which uses a conjugate gradient method in conjunction with the previously mentioned random search methods. A different random method is described by Hu et al. [107].

Simulated Annealing (SA) has had a number of applications to MLP training. Sun et al. [228] propose a random optimization method, based on deterministic annealing. Rosen and Goodwin implement simulated annealing and its variants (fast simulated annealing and very fast simulated annealing) [200]. Chen and Sheu compare a number of algorithms, including a simulated Cauchy annealing algorithm, on a 1-D sine wave approximation problem [43]. A further implementation of SA is reported by Porto [179]. Boese and Kahng [37, 36] consider optimal annealing schedules for a 4-4-1 MLP on a subsoil object detection problem. Boese suggests that while simulated annealing is slower than other methods, it remains useful for finding points of lowest error.

Algorithms which involve hybrid local-global schemes have also been proposed, such as the deterministic method of Tang and Koehler [231] and Orsier's [168] algorithm (incorporating a random line search, scaled conjugate gradient and a 1-dimensional minimization algorithm). Empirical results show improved performance of Orsier's method over the above-mentioned algorithms of Baba et al. [8, 9] and Tang and Koehler [231]. Other hybrid algorithms include the tunneling method of Barhan et al. [21, 217] and Treadgold's method which combines simulated annealing, weight decay and RProp [238].

Finally, Evolutionary Algorithms (EA's) have been used extensively to train MLP's [39, 257]. This includes algorithms which modify the topology and/or connectivity of the network

as part of the learning process. Many EA's use a binary encoding representation of the weight vectors. Similarly, Battiti and Tecchiolli have applied the Reactive Tabu Search algorithm [27], which also uses a binary representation. Evolutionary optimization methods are discussed further in Chapter 5.

2.7.3 Further Research in MLP Training Algorithms

It is important to note that research continues to produce new MLP training algorithms, which display impressive performance in the available experimental results. This research continues to benefit from previous work and the application of knowledge from other fields of research. These approaches incorporate Bayesian methods [160], homotopic methods [84, 217], stochastic algorithms [139], minimum description length [99, 100, 101], constrained optimization [123], and other methods [243, 71, 156, 172, 208].

2.7.4 Practical Training Heuristics and Prior Knowledge in MLP Optimization

The formulation of MLP training as an optimization problem, as discussed above, is developed by regarding the MLP as a "black-box" model, that is a device that can only be manipulated by adjusting the given variable parameters. No prior information concerning the MLP model or the supervised learning problem is incorporated into the problem statement. For example, backpropagation is simply the implementation of gradient descent over a space of variable parameters (weights).

Many of the algorithms mentioned above make assumptions or incorporate heuristic knowledge in an attempt to provide effective learning. Progress in research has led to an overall more principled approach to the incorporation of this knowledge, as can be seen from the algorithms mentioned in Section 2.7.3. In general, this trend has also meant that prior knowledge has been used more extensively, and the resulting algorithms are more complex than simple optimization methods.

Prior knowledge has been incorporated into MLP training in another, perhaps less obvious manner. No MLP training method is completely autonomous; implementing a training method typically involves choosing various constants, algorithm parameters and so on. As an example,

using the backpropagation algorithm involves choosing a value for the learning rate, a method for initializing the weights prior to training and an appropriate activation function for hidden and output units. This process can be considered a "meta-optimization" problem in itself, which is most often solved by trial and error on the part of the user.

Since the proposal of backpropagation, practitioners have developed a number of heuristics which are commonly used when training an MLP. These heuristics have been gained from principles of applied statistics, through practical experience and by trial-and-error. Although heuristics have been misleading and sometimes only work in certain circumstances, a number of them are widely considered to be very useful for improving the training speed and quality of solution reached by training algorithms [138].

1. *Initialization of Weight Values*. Rumelhart et al. [203] suggested that the weights in an MLP should be initialized to small, random values prior to training. Since then many researchers have investigated the effect of the initialization method on the training performance (for a summary see [234]). The performance of backpropagation and other algorithms can be highly dependent on the initialization of weights [130, 173].

   The general heuristic commonly used is to initialize weights such that the sigmoidal activation functions in the network operate in their linear range [138] (very small weights however result in very small gradient values). This issue is not of direct consequence for algorithms which do not use gradient information, but has implications for the structure of the error surface. This relationship between weight initialization and error surface is discussed further in Section 4.2.4.

2. *Pre-processing of training data*. In any supervised learning task, the data is usually subject to some kind of pre-processing, prior to training. This pre-processing includes dimensionality reduction techniques, which attempt to reduce the number of inputs without losing information relevant for training. Some general guidelines include the removal of any biases in the inputs by translating such that (over the training set) their mean values are close to zero and their variances are similar to each other [138]. A motivation again is to (on average) use the full useful operating region of the sigmoidal functions. A further step is to remove correlations between inputs.

3. *Choice of Error Function*. Although SSE and MSE are used in this thesis, a number of

different error functions have been used in MLP training, depending on the data and the interpretation of the trained network's output values (see [32], Chapter 6). Different error functions will lead to different error surfaces, though the basis of most error functions is the same, that is, measuring performance of the network on a set of training data. Intuitively it might then be expected that these surfaces would still have many similarities (e.g, location of minima), with the different functions accentuating or smoothing such features.

4. *Learning Rate*. Standard backpropagation uses a single learning rate parameter, η. The value of the learning rate is problem dependent and is usually set by the practitioner during training experiments. In effect, this parameter constrains the step sizes that are taken in each component direction in weight space through the multiplication with a common constant. Many other local and global algorithms incorporate one or more parameters which, like backprop's η, govern the step size taken over the error surface. Principled methods exist for computing the single optimal learning rate for backpropagation and optimal learning rates for each weight, using Hessian information during training [138].

5. *On-line methods*. On-line or stochastic algorithms have proven to be effective training methods. In the case of redundancy in the dataset, on-line methods result in a significant decrease in expected training time [138].

Practical Experience

It is possible to explain the respectable performance of backpropagation and other algorithms in training MLP's. Backpropagation is a simple gradient descent technique and *not* some powerful algorithm which is able to solve the training optimization problem very efficiently and accurately. In fact, there are several factors which help to explain the success of backpropagation:

1. MLP's are known to be capable of impressive representational power, both in theory (Section 2.4.3) and in practical situations (training and testing performance, structure of decision boundaries). This power can be partly attributed to the typically large number of free parameters which are used in an MLP, together with methods for limiting the flexibility of the model to produce solutions which provide good generalization [32].

This ability means that backpropagation at least has the possibility of finding reasonable solutions to a broad class of learning problems (compared to, e.g. a linear model, which is guaranteed to perform poorly on highly non-linear data).

2. There is some evidence to suggest that the error surface is often "well-enough" suited to gradient descent that backprop performs acceptably.

3. The practitioners of backpropagation make many heuristic choices in implementing the algorithm, and making these choices involves repetition and refinement of experimental settings.

4. The training data is often subject to pre-processing prior to training.

It is worth remembering that different training algorithms will typically explore the error surface differently. Hence, an algorithm which performs worse than another for a given problem, from a given starting point, encounters different error surface characteristics as a result of its dynamical behaviour.

2.8 No Free Lunch - the Nature of Optimization Problems

The process of *training* an MLP on a set of input-output data, to perform a classification, function approximation or other mapping task, is clearly an optimization problem. Two questions arise which are essentially (but perhaps deceptively) similar:

- What can be said about "how good" an MLP is for learning these mapping tasks; more generally, given a set of data to learn, is an MLP the best model for learning to approximate the data[6], or is some other model preferable?

- What can be said about using different optimization methods for training an MLP - which one is best? Even more generally, are some optimization algorithms superior for training MLP's than others?

These general questions are concerned with the nature of optimization problems themselves.

[6] Although generalization is usually the goal of training, the simpler goal of minimizing an error function over a finite training set is considered here.

Recent research by Wolpert and Macready has directly addressed these issues [147, 255]. The resulting "No Free Lunch" (NFL) Theorems have had considerable impact on the field of optimization in complex systems such as in the development of training algorithms for MLP's.

Given an optimization problem, the ultimate objective is to apply an algorithm which finds the global optimum reliably and quickly. In this sense a problem is easy if the algorithm in question can achieve this objective, otherwise it is hard. Is there some class of problems (cost functions) that is intrinsically harder than others, *independent* of the algorithm in question? Macready and Wolpert show that, when averaged over all cost functions, all algorithms can be expected to perform *equally* well. They consider the histogram, $\mathbf{c}$, of cost values that an algorithm, a, obtains on some cost function, f, in m distinct cost evaluations. Any measure of hardness of a cost function or performance of a search algorithm can be defined from $\mathbf{c}$. By summing over the set of all f, the theorem shows that

$$\sum_f P(\mathbf{c}|f, m, a_1) = \sum_f P(\mathbf{c}|f, m, a_2)$$

that is, the conditional probability that histogram $\mathbf{c}$ will be obtained under m iterations of algorithm a_1 is equal to the probability that $\mathbf{c}$ will be obtained under m iterations of algorithm a_2. The corollary is that for any performance measure $G(\mathbf{c})$, the average over all f of $P(G(\mathbf{c})|f, m, a)$ is independent of a.

The theorem is proven for deterministic algorithms which do not revisit points in the search space, however it is shown that the theorem also applies when considering stochastic algorithms and algorithms which revisit points.

In short, the NFL theorems state that

> *... all algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions.* (Wolpert and Macready [255], p. 1)

It is also noted that there may be "minimax" distinctions between algorithms, which NFL does not address directly. Since a_1 and a_2 perform equally on average over all f, it might be the case that there are a small number of cost functions for which a_1 performs much better than a_2,

but there are also a large number of cost functions for which a_2 performs only slightly better than a_1. In this sense, there is a "head-to-head" distinction between a_1 and a_2.

A further NFL result shows that the performance of any algorithm on some cost function up to some point in time gives no indication of its performance on that cost function in the future.

While the NFL theorems have little direct practical relevance, the underlying message behind them certainly does. When solving an optimization problem, the assumptions made (both explicit and implicit) and the prior knowledge used are paramount

> *In short, there are no "free lunches" for effective optimization; any algorithm performs only as well as the knowledge concerning the cost function put into the cost algorithm.*(Wolpert and Macready [255], p. 2)

Often, many assumptions are made in application. Although the NFL theorems do not hold in this case, the class of functions may still remain very large. Consider making some smoothness assumption about the error surface, or choosing the form of the cost function itself in training an MLP (e.g. the MSE function). Many algorithms may still perform equally given such assumptions, but others may be less suited to these restrictions. It is an important and relatively open area of research to move towards understanding the magnitude of problem spaces and the effects that various assumptions have on reducing such spaces.

In areas of research such as learning algorithms for MLP's, the focus has typically been on designing effective algorithms for optimizing the weights of an MLP, often without incorporating prior knowledge concerning the particulars of the network, the training set, etc. The algorithms are simply applied to the task as they would be to any other task which can be cast as an optimization problem. The NFL results suggest that the primary approach should be to "start with the given f, determine certain salient features of it, and then construct a search algorithm, a, specifically tailored to match those features" [255]. What is typically done is the inverse - "to investigate how specific algorithms perform on different f's".

> *Ultimately, the only way to justify one's search algorithm is to argue in favour of a particular $P(f)$, and then argue that your algorithm is well suited to that $P(f)$. This is the only (!) legitimate way of defending a particular search algorithm against the implications of the NFL theorems.* (Wolpert and Macready [255], p. 25)

For good performance, the dynamics of the search should be well-matched to the nature of the cost function.

2.8.1 Discussion

This section together with Section 2.7 above has been concerned with the question of the search for an optimal algorithm or method for solving the MLP training optimization problem. The NFL theorems indicate that without the incorporation of

- assumptions regarding the nature of the error surface, and/or
- prior knowledge specific to the problem

all algorithms can be expected to perform equally, averaged over all problems. However Section 2.7 shows that assumptions and prior knowledge *are* in fact an integral part of most of the training algorithms which have been developed and applied to MLP training. The main difficulty is that the assumptions and prior knowledge have been incorporated with little regard for their implications for training performance. The importance of the NFL results is to emphasize that the relationship between the training algorithm, the assumptions and prior knowledge used and the error surface are crucial to MLP training tasks. This problem has been little addressed in the literature. Part of the aim of this thesis is to provide techniques that facilitate the exploration of this relationship.

2.9 Comparing Different MLP Training Methods

Given the large amount of research into developing better MLP training algorithms, the issue of comparing the performance of different algorithms becomes important. At a higher level, practitioners are also interested in comparing the performance of different supervised learning methods. An MLP, trained using one of many particular algorithms is a perfect example of this.

Unfortunately, comparative studies between different training algorithms have often been less than adequate [161, 183]. New algorithms have often been presented with insufficient detail required to reproduce the author's results. For example, the kind of heuristic decisions made during implementation (Section 2.7.4) are sometimes not reported. As a result, these algorithms

remain little used and have not been subject to independent testing. Another problem is the lack of sufficient statistics of multiple trials of algorithms.

There are difficulties associated with some of the datasets that have been used to compare and test training algorithms. XOR is perhaps the most commonly used training task, but contains only 4 training patterns and despite its simplicity is implemented in different ways (e.g. binary (0,1) v's bipolar (-1,+1) training patterns and unit activation ranges). Prechelt [182] reports from a study of 113 articles published in journals in 1993 and 1994, that only 1 in 3 uses even a single real-world dataset for testing algorithms, and only 6% use more than one such dataset. The particular training tasks chosen also tend to vary widely throughout the literature.

When these problems with comparative studies are considered together with the implications of the NFL theorems, it is not surprising to find that the results of comparative studies are sometimes contradictory or inconclusive. Some examples of good comparative studies can be found in [43, 74, 184, 192, 200, 217]. Some of the more powerful algorithms based on techniques from optimization (i.e. CG, LM and QN methods) are often considered to be among the best choices for training [51]. Lawrence however, argues and gives evidence that such methods have sometimes given poor results, and in particular may not be scale up to be suitable for problems involving large networks (see [137] and the references therein). Lehr also reports a lack of success with these methods, and suggests that on-line backpropagation often provides better performance in the case of large problems [139]. Lehr goes on to develop a number of stochastic-update algorithms which aim to improve on on-line backprop. Other comparative empirical studies can be found in most of the references to algorithms in this section. Fahlman [74] and Neal [161] provide discussion of MLP benchmarking issues, and Dietterich [66] discusses statistical tests for comparing algorithm performance.

It is apparent that empirical studies support the overall conclusion that it is not possible to recommend one particular method, and suggest trying a number of different algorithms for any given task [115]. Popular choices include the CG, LM, QN and stochastic methods. On-line backpropagation often produces good results, hence backpropagation remains widely-used, aided by its simplicity and relative computational efficiency.

The development of training algorithms has led to numerous improvements in the solution quality and training time of MLP's on a wide variety of learning problems. For the practitioner, the question of which algorithm should be used for some given learning problem is

unfortunately not answered by existing analytical or empirical results. A number of studies have concluded that several algorithms should be tested on a given problem, to provide the best results. The body of algorithms is too large to allow a meaningful empirical study comparing the performance of all algorithms, and many of these algorithms remain poorly tested and not widely used.

2.10 Summary

This chapter has described the application of MLP neural networks to supervised learning problems. The error surface is defined by the formulation of the training task as an optimization problem over the space of weights in the network. Following chapters develop methods for understanding the nature of MLP error surfaces and the dynamics of learning algorithms on the error surface.

It is clear that despite the many hundreds of algorithms that have been developed for training MLP's, it is not possible to provide a single algorithm which offers superior performance. This partly stems from the NFL theorems and partly from the lack of careful, extensive testing of all these algorithms in practical training situations. The effectiveness of a training algorithm is dependent on the structure of the error surface, and therefore dependent on the particular problem and data used. Although the NFL theorems suggest that all algorithms are equal on average, typically prior knowledge and assumptions are an integral part of the application to an MLP training algorithm. In this case the NFL results do not apply. The prior knowledge and assumptions can change the structure of the error surface through manipulation of the data and network model, as well as changing the location and dynamics of the algorithm on the error surface.

This thesis takes a different approach to the MLP training task. The central focus is not on the solution to this black-box problem but rather on the problem itself. It is through a better understanding of the nature of the problem that prior knowledge and heuristics can be combined with powerful optimization algorithms to provide improved methods for solving the problem. The optimization problem and black-box formulation of training is merely a starting point from which the specifics and unique qualities of this particular optimization task must be determined. In addition, understanding the nature of the problem will result in a deeper understanding of

the performance of different algorithms, and allow more meaningful comparisons in practical situations.

Chapter 3

Visualization and Neural Network Learning

This chapter is concerned with the use of scientific visualization methods for the analysis of neural networks, in particular Multi-Layer Perceptrons (MLP's), the training process and the error surface. The motivations and methods are discussed, and the task of visualizing multivariate data is highlighted. After discussing various approaches to visualization of learning and representation in MLP's, a method is described using the well-known statistical technique of Principal Component Analysis. This is found to be an effective and useful method of visualizing the learning trajectories of many learning algorithms such as backpropagation, and can also be used to provide insight into the learning process and the nature of the error surface.

3.1 Scientific Visualization

One very general and important issue in many areas of science is the manipulation of large quantities of numerical data. The process of scientific exploration leading to a scientific hypothesis, relies on gaining insight from the exploration process. *Scientific visualization* is one method of exploratory data analysis. Scientific visualization is generally defined as the process of representing data in graphical or visual form, for the purposes of descriptive analysis (e.g. detecting features or structure in the data such as trends, regularities, anomalies and outliers). The advances in computer technology in the last few decades have permitted significant advancement of scientific visualization, allowing rapid display and processing of data, as well

as high quality graphical displays for detailed visualization, animation and simulation of data. As computational speed and capacity continue to increase, so do the possibilities for scientific visualization.

Humans have highly developed visual pattern recognition abilities, and are able to synthesize *information* from visual data in a much more effective and efficient manner than from examining large lists or tables of numbers. Scientific visualization is thus a natural means of using the power of the eye-brain processing system for data exploration [235].

This thesis focuses on several general properties which are widely considered to be desirable for visualization techniques. These properties are:

- simplicity
- flexibility
- interpretability
- clarity

Chambers et al. [40] suggest that a *flexible* method is one that is useful on a wide variety of data, as well as being somewhat insensitive to the quantity or quality of data. An *interpretable* method may require some time to become familiar with, but should still be usable. *Simplicity* is stressed as an important factor in achieving these two things without requiring excessive user effort. Methods should also allow iteration (successive application, interaction and refinement) as exploratory analysis is inherently an iterative process. Finally, the meaning of a visualization method should be clear, and should not be open to different interpretations from different user's. It is important to consider the interaction between the user and the visualization technique, to ensure ease of use and the ability to customize the technique in response to the changing needs of the user [50].

3.2 Types of Data

One possible categorization of the different types of data available for visualization and statistical methods is shown in Table 3.1. Consider a single variable, x, which is taken as the representation of some given quantity of interest. If x is a qualitative variate, then observations of x take on one of a fixed number of quantities, which may be ordered (e.g. $x' \in$

Qualitative	ordered
	unordered
Quantitative	discrete
	continuous

Table 3.1: Categorization of data.

{cold,cool,warm,hot}) or unordered (e.g. $x' \in$ {red,green,blue}). x may also be a quantitative variate, in which case observations may be either discrete (finite or countably infinite; e.g. the roll of a die) or continuous (uncountably infinite; e.g. a sample drawn from a Normal distribution $\mathcal{N}(0,1)$). This chapter is mostly concerned with the visualization of continuous, real-valued data, $\mathbf{x} = (x_1, \ldots, x_N) \in I\!R^N$.

3.3 Visualization of Multivariate Data

The visualization and exploration of data has long been studied, and has come to be known as Exploratory Data Analysis (EDA) [239]. While the techniques available are many (see e.g. [40, 46, 187]), the visual focus of EDA has been primarily concerned with the investigation of data given the constraints of one, two and three dimensional graphical displays. If the data consists of observations of just a single variable, qualitative data is often represented as a bar chart or pie chart, whilst quantitative data can be described by a frequency histogram. Broadly speaking, one and two dimensional data can be visualized directly on a computer screen, by plotting each data point on the Euclidean axes (see e.g. Figure 3.1). This method provides a simple, understandable picture of the data that gives the user an indication of features of the data such as clustering, outliers, and the overall spread of the data. Three dimensional data can be similarly represented on a two-dimensional screen or piece of paper, using non-orthogonal axes and relying on the observers concept of perspective and experience in a three dimensional environment. A significant amount of work in this area has resulted in software capable of providing animated rotation and manipulation of three-dimensional graphics, solving the problem of the "hidden" view of the 3-D data.

Although the dimensionality of such data does not present a fundamental problem for visualization, there are other difficulties that require attention. An important one is related to precision and scale: as the number of data points increases it becomes difficult to distinguish different points. Methods of summarizing the nature of the data, such as computing derived

statistical quantities such as mean and variance (which are often based on assumptions about the data), are partly developed to address this issue [46]. This thesis concentrates on the special issues and problems of the visualization of multivariate data, notwithstanding the general applicability of the principles of EDA to multivariate visualization [98].

In general, multivariate data may consist of a number p of observations or data samples of a number N of different variables. Such a dataset can be represented as a data matrix, $\mathbf{X}$

$$\mathbf{X} = \begin{pmatrix} x_1^1 & x_2^1 & \dots & x_N^1 \\ x_1^2 & & & \\ \vdots & & & \\ x_1^p & & & x_N^p \end{pmatrix}$$

where x_i^j denotes the jth sample of the ith variable. The task of analyzing $\mathbf{X}$ is to examine relationships between any of the variables and/or any of the samples. Often, prior knowledge or assumptions can be used to restrict this task [133]. If it is known that the data are drawn from a number of *groups*, then analysis is focused on this grouping of either the rows or columns of $\mathbf{X}$ and the relationship between the variables (columns) and this grouping structure. For example, if the data consists of sample vectors of handwritten numeral features, it is known that the data belongs to one of ten classes (i.e. $0, \dots, 9$). This knowledge might mean that analysis is focused on the columns of $\mathbf{X}$, to explore how well each feature (column) of the data allows the different classes to be distinguished. Alternatively, if no class labels are known, it may be of interest to attempt to group or cluster the data and investigate the relationship between this clustering and the different data samples (rows) of $\mathbf{X}$, since this relationship may reveal trends or anomalies in the manner in which the data was collected.

For data of higher dimensionality than three, the difficulty of visualizing the data (in particular, the relationships and interactions between variables) is greatly increased, since humans have no physical, geometric notion or experience of such spaces. Various methods have been developed to address the problem of multivariate data visualization, some of which are summarized here.

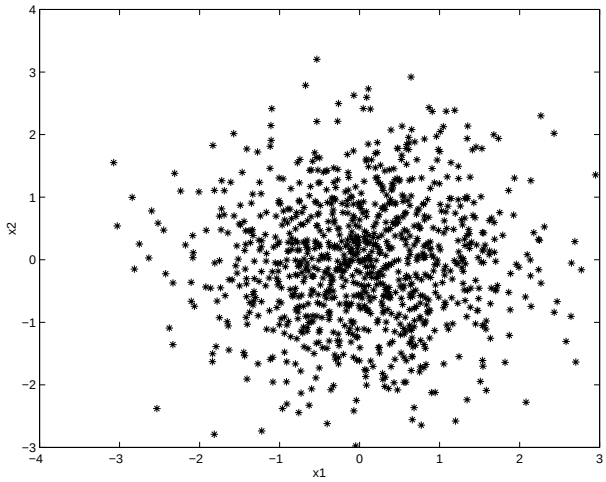

Figure 3.1: 1000 data points with x_1, x_2 drawn from independent $\mathcal{N}$(0,1) distributions.

3.3.1 Displaying Many Variables Simultaneously

Draftsman's Display

Perhaps the most straightforward means of displaying multivariate data is to produce many different displays or plots of all pairs of variables simultaneously, referred to as a scattergram, pairwise scatter diagram or Draftsman's Display [149]. Figure 3.2 shows a Draftsman's display for the thyroid dataset [34]. Scatterplots for each pair of variables lead to 10 plots for this dataset of 5 variables (and to $\frac{1}{2}(N^2 - N)$ plots for N variables).

Such an approach is simple and does not intrinsically result in the loss of information associated with a dimensionality reduction. Unfortunately, the number of plots required means that the method becomes unmanageable for N greater than about 10 variables. In addition, the structure present in the N-dimensional data is often difficult to discern from the plots of pairs of variables, partly because each variable appears in $(N - 1)$ plots.

If plotting the complete draftsman's display is infeasible, some group of scatterplots may be plotted. This visualization can be particularly useful in exploring the relationship between a single variable (which is plotted on the y axis on each scatterplot) and a number of other variables (one scatterplot for each, with this variable given along the x axis; see, e.g. [46, 47]). Scatterplot techniques have found wide application in the fields of machine learning and pattern recognition (see, e.g. [194]).

A related approach used in the neural network domain by Lawrence [137] when the dimensionality of the data is very high is to plot a set of two-dimensional/variable sub-plots, where

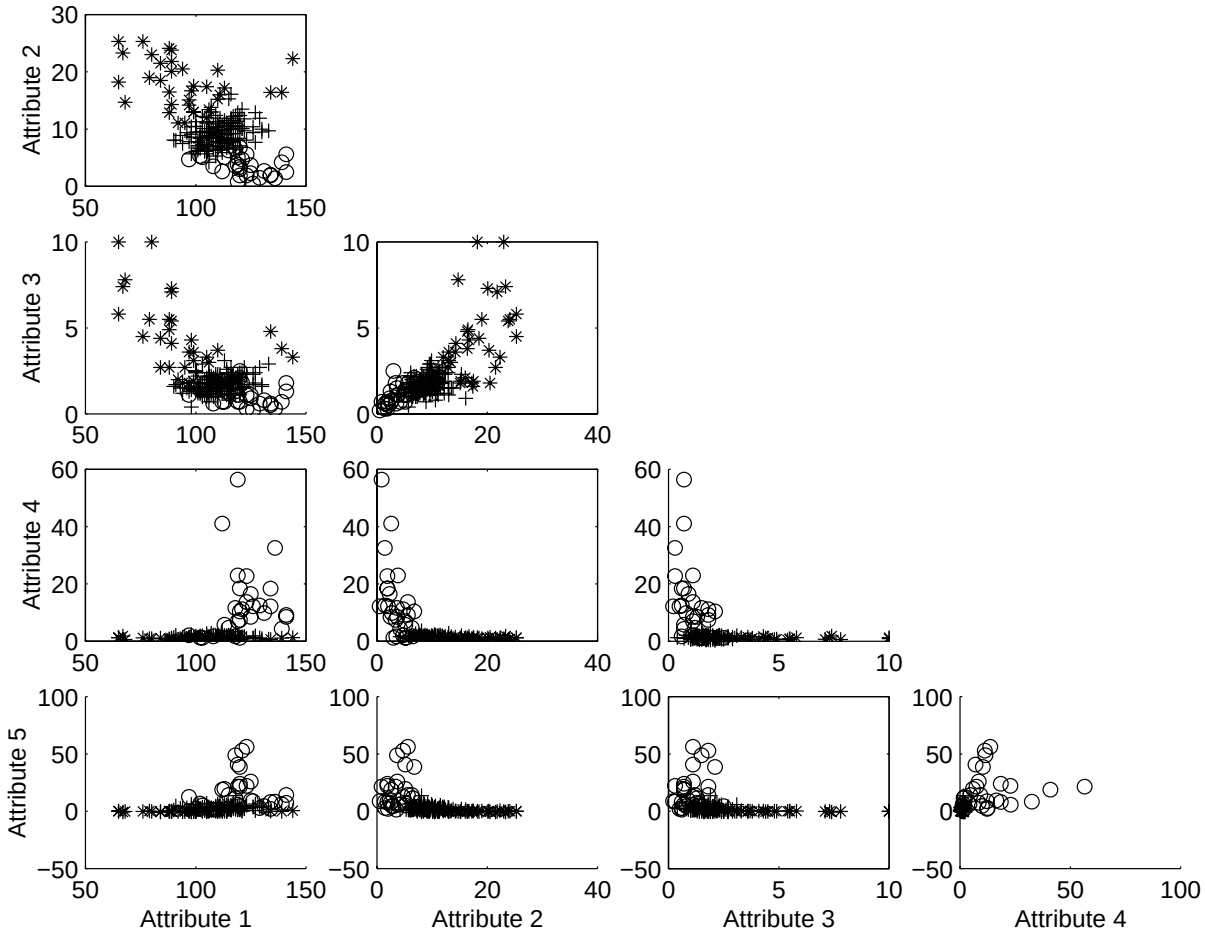

Figure 3.2: Draftsman's Display or pairwise scatter diagram of thyroid data.

each sub-plot is chosen randomly from the set of all possible similar projections. This technique is used to examine the input-output function produced by an MLP, as well as to examine the error surface (see Chapter 4).

Glyph Scatterplots

A different approach which is also well known involves representing each observation in the sample as an adjustable symbol, known as a glyph [72, 133]. A simple example is to represent three-dimensional data samples by a circle, with the first two dimensions specifying the center of the circle, and the third the radius length of the circle. Another example is to represent the third and fourth dimensions of a data sample by a pair of axes emerging from each point plotted in the usual 2-D plane, with the length of these axes indicating the magnitude of the values of each sample in those dimensions. An example of this method is shown in Figure 3.3. When quantitative data are to be represented in this way, there must be a means of representing

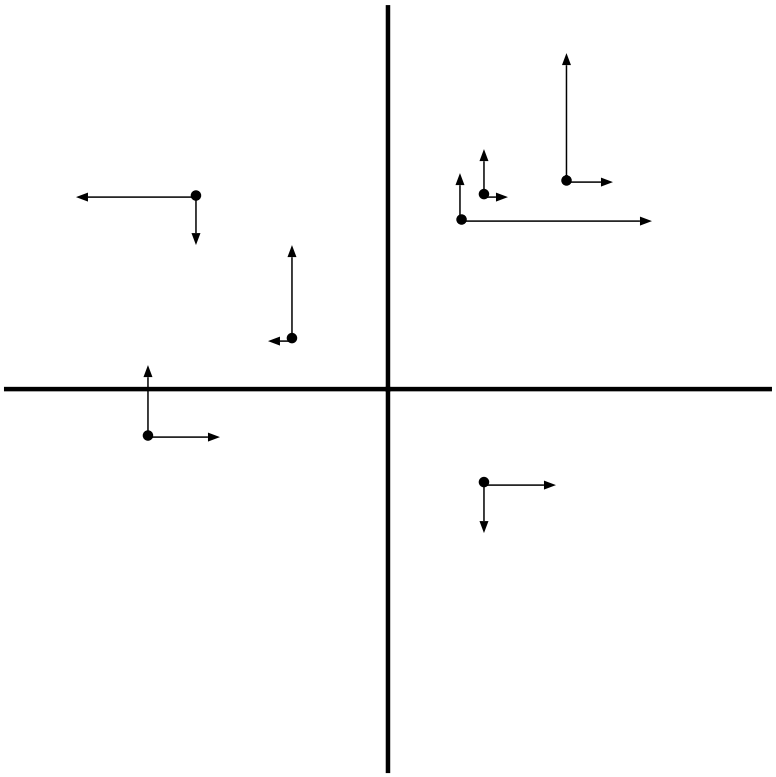

Figure 3.3: Axes glyphs used to display artificial four-dimensional data.

negative values. In Figure 3.3, axes in the north and east directions indicate positive values, while axes in the south and west directions indicate negative values.

The advantage of using glyphs is that the symbol itself can be used to represent the values of some of the variables in each data point, thus conveying some useful information. High quality computer displays open further possibilities of using other visual attributes such as colour and brightness to encode variables.

This kind of visualization is most often used when the sample contains qualitative variables, since it is easier to interpret such quantities as features of glyphs compared to a continuous quantity. Clearly the technique is limited to data of dimensionality about 5 or 6, as the display becomes increasingly difficult to interpret.

Chernoff faces

A variation on the use of glyphs for data representation is to remove the two dimensional axes and represent all dimensions of the data through features of a composite, familiar object. Perhaps the most well-known method of this kind is known as Chernoff Faces, being first suggested by H. Chernoff in 1973 (see [133]). Each data point is represented by a cartoon image of a hu-

man face, with different features of the face varying according to the values of the different dimensions in the data point. The choice of a face representation is motivated by the large amount of experience people have in studying faces, as opposed to other objects.

There are two major difficulties with this method of data visualization. The first is that different features of the face may convey more information than others, because of their size and shape (and how they vary). The second is that different observers may be subjective in comparing the features of different faces, leading to different interpretations of the same data. These issues are compounded as the number of data points/faces increases, decreasing the effectiveness of the technique. As with glyph plots, qualitative data may be easier to interpret as a Chernoff face, as opposed to continuous data.

The Chernoff Faces technique has previously been applied to the visualization of the MLP, as well as animating the backpropagation training process by updating the face display during learning (e.g. after each epoch) [240]. The effectiveness of the application in general reflects the issues discussed above. It is suggested that the technique may be effective as an educational tool in neural network principles.

Andrew's curves and functional representations

It should be clear that multivariate data may be represented in a number of different ways, in order to allow complete visualization in three or less dimensions. While glyph scatterplots transform each data point into a pictorial or geometrical symbol whose properties reflect the values of the variables of each data point, a functional representation may be considered which maps a multivariate data set onto $I\!R^1$. Andrews' plots are the most well-known technique of this kind [117, 149]. For each observation, $\mathbf{x}$, a function is defined

$$f_{\mathbf{x}}(t) = \frac{\mathbf{x}_1}{\sqrt{2}} + \mathbf{x}_2 \sin(t) + \mathbf{x}_3 \cos(t) + \mathbf{x}_4 \sin(2t) + \mathbf{x}_5 \cos(2t) + \ldots, -\pi \leq t \leq \pi$$

These functions have the property that points that lie close together in the data space will produce curves that remain close together for all values of t, while more distant points in the data space will produce curves that are distant for all t - preserving Euclidean distances, means and variances of the original data.

The Andrews' Curves technique is illustrated here through its application to the visualization of MLP multivariate weight data, which was generated in two different ways. Firstly, a 2-2-1

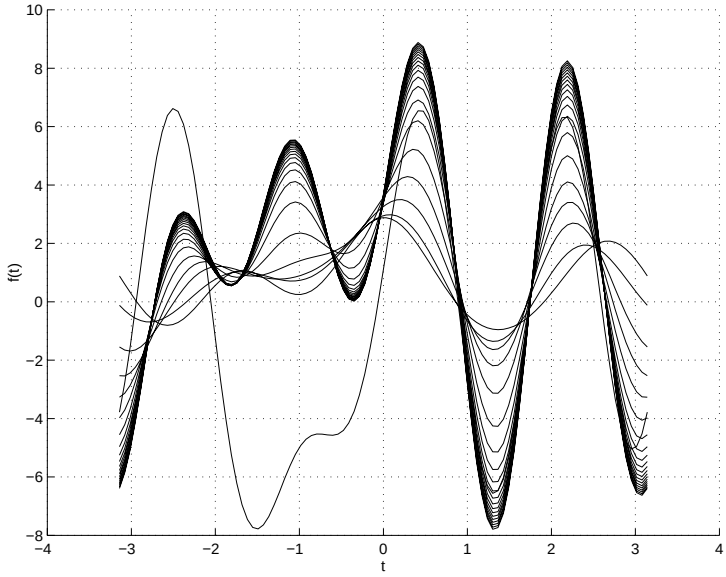

Figure 3.4: Andrews' curves for data sampled at regular intervals during a backpropagation training run, $\eta = 0.2$.

MLP net was trained using standard backpropagation on the Exclusive-OR (XOR) problem. Batch learning was used, with $\tanh()$ activation functions at the hidden and output units. The networks were trained for 1000 epochs, and the weight values recorded at the commencement of training and after every 50 epochs. This sample, consisting of 21, 9-dimensional data points, was then visualized using Andrews' Curves. Figure 3.4 shows a sample collected for $\eta = 0.2$. Training progresses from the initial curve (which is very dissimilar in form to all the other curves), towards the dense cluster of curves observable in this Figure. Thus training is captured as the morphing of a curve from an initial curve to a final curve. The rapid transition at the start of training, due to large initial error, can be reduced by reducing the learning rate. Figure 3.5 shows another training run for $\eta = 0.01$, which converges more slowly towards a solution weight vector. Comparing Figures 3.4 and 3.5, it is clear that the two different training runs have led to rather different locations in weight space. Note that this is a slightly unusual data set compared to the multivariate data commonly used for Andrews' curves. The MLP training data is in fact a trajectory (i.e. a *time series*), which represents a record of the dynamics of the backpropagation training algorithms on this training problem. Such data is often the focus of interest in this dissertation.

In the second instance, data samples were obtained from the same MLP architecture and training problem. This time however, data points were collected by sampling from the multivariate Normal distribution $\mathcal{N}(0, 1)$, centered on the origin in the $\mathbb{R}^9$ weight space. Figure 3.6

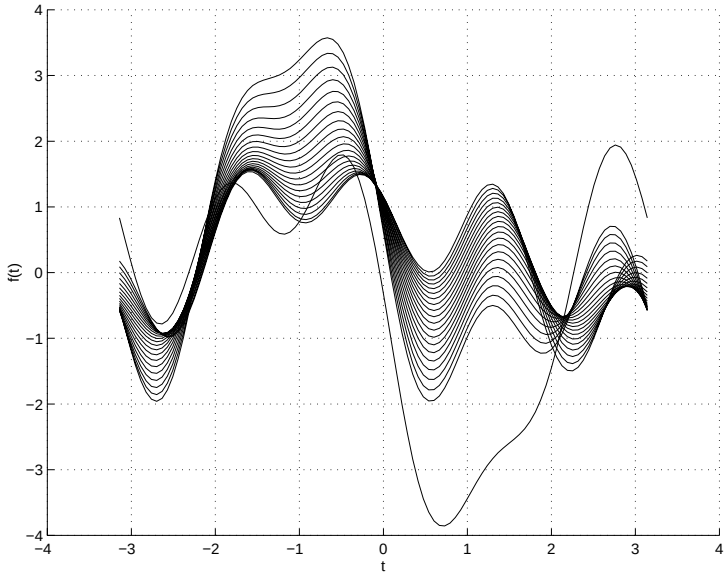

Figure 3.5: Andrews' curves for data sampled at regular intervals during a backpropagation training run, $\eta = 0.01$.

shows Andrews' Curves for a sample of 25 random weight vectors. As might be expected, the sample appears quite random - no features are immediately observable. This sampling was repeated with a rejection criterion applied to the samples generated. Samples were rejected on the basis of the corresponding values of the mean-squared error (MSE) function on the XOR training set. The error value below which points were accepted into the data set was adjusted by hand until the ratio of accepted to rejected points was $\approx 1 : 5 \times 10^5$. This ratio corresponds to a threshold MSE value ≈ 0.77. The result is shown in Figure 3.7 for a set of 25 samples. It can be seen that several clusters are present in the data set, as indicated by the groups of tightly coupled curves. The indication is that these "low-lying" points are not uniformly distributed but are to some extent clustered into certain regions in different locations on the error surface.

Andrews' Curves are most applicable to data sets containing a fairly small number of observations (~ 50), since the visualization becomes crowded with curves for larger data sets. In addition, the technique is also well-suited to problems of dimensionality up to around $N = 20$ - increasing the dimensionality introduces increasingly higher frequencies into the curves, making the detection of clusters or features more difficult to discern. Representing different dimensions by different frequencies may also lead to the problems of interpretability, as discussed for Chernoff Faces above.

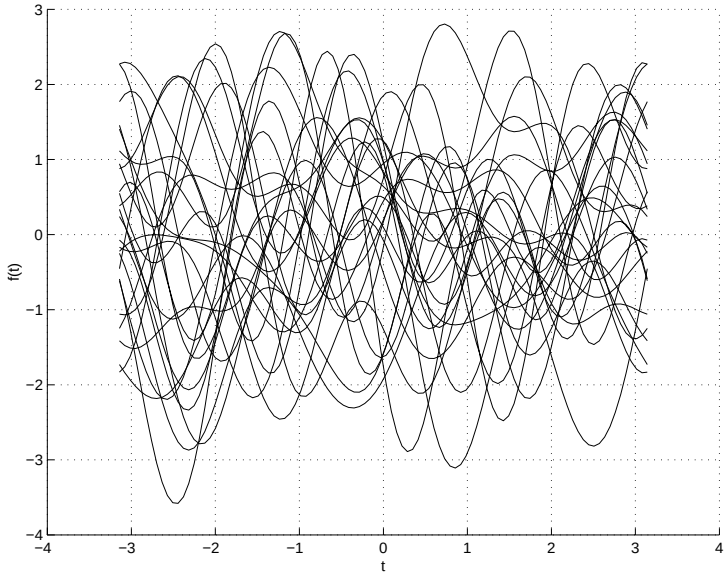

Figure 3.6: Andrews' curves for data collected by sampling from weight space of a 2-2-1 MLP, given the XOR task, from a multivariate $\mathcal{N}(0, 1)$ distribution.

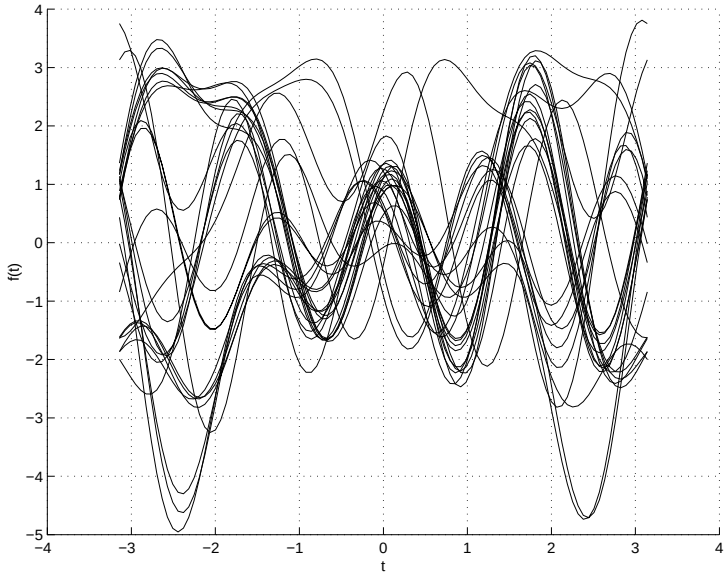

Figure 3.7: Andrews' curves for data collected by sampling from weight space of a 2-2-1 MLP, given the XOR task, from a multivariate $\mathcal{N}(0, 1)$ distribution. Only 1 in $\approx 5 \times 10^5$ points are accepted, according to their corresponding fitness values.

3.3.2 Subspace and Dimensionality Reduction Visualization Methods

The above methods aim to provide a visualization of multivariate data through a transformation that facilitates direct display of that data. A different approach is to project the data onto a space of lower dimensionality, so that it can be more easily displayed, whilst at the same time minimizing the loss of information associated with making such a projection. Various techniques of this kind are commonly referred to as subspace visualization methods. Subspace methods basically involve:

1. Finding an optimal representation of the density distribution of the multivariate data;
2. Finding a low-dimensional presentation of this representation that preserves the most important features of the data (e.g. distances, or some other criteria).

Terekhina provides a review of early research in this area [232].

A projection of some multivariate data point $\mathbf{x}$ is said to be *linear* if it can be written in the form

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

where $\mathbf{y}$ is the resulting (projected) vector and $\mathbf{A}$ is a real transformation matrix. This projection can be viewed as a rotation of the coordinate system. To reduce the dimensionality, it is necessary to replace some of the basis vectors with constant values, thereby projecting the data into a subspace of the original space (see Section 3.5).

Linear projection methods have the advantage of being computationally simple. On the other hand, non-linear methods offer the possibility of producing a projection that results in a smaller loss of information, at the cost of increased computational complexity. Non-linear methods are usually implemented through iterative numerical algorithms that seek to optimize some given cost function, which corresponds to a non-linear optimization problem [133]. These issues are closely related to those of bias-variance and model complexity mentioned in Section 2.4.2. This is one example of the tradeoff between the flexibility and computational complexity of statistical modelling methods [32].

One of the most well-known techniques for reducing the dimensionality of a set of data is Principal Component Analysis (PCA), which is discussed in detail in Section 3.5 below.

The *dependencies* within a set of multivariate data are of major interest from the visualization point of view - if the variables are statistically independent, the data can be decomposed, and univariate visualization methods are applicable. The number of data samples must of course at least equal the number of variables n, since n data points can span at most an n-dimensional basis. In practical situations, the dependencies in a multivariate data set may be non-linear and hence non-trivial to detect.

It is often the case that the goal of multivariate data analysis is to discover groups or clusters within the data (for an overview of clustering methods, see [73]). Clustering methods typically cluster the data into a specified number of groups, according to some dissimilarity measure. Clustering methods can in addition be viewed as examples of *unsupervised learning* methods, since data samples provided for clustering have no correct output or label associated with them. Unsupervised methods are however often used to solve problems similar to supervised learning [194], for example solving a classification problem by clustering the input data vectors (the number of classes is known and is set equal to the number of clusters), and assigning the output vector for an unseen input as the cluster which best fits the data point. While the intention is not visualization as such, clustering often provides a convenient representation of the data for visualization.

3.3.3 Current Research in Multivariate Visualization

Scientific visualization of multivariate data is a field which is gaining increasing attention in research. Advancements in computer processing speed, graphics capabilities and other hardware have allowed researchers to develop methods involving animation, visualization of complex phenomena such as fluid flow, interactive displays involving virtual reality and 3-D environments and the ability to provide detailed multiple views and projections of data in real or minimal time [186]. These developments have also meant a large increase in the amount of data (in terms of numbers of variables and samples) which can be collected and analyzed.

Software has been developed which utilizes special-purpose computer hardware to facilitate the interactive exploration of six-dimensional data [258]. The data is represented in a 3-D "visual space", which is a projection of the data. The user controls the visual space, altering the projection smoothly via operators such as rotation, providing a "visual" tour of the data. For data of very high dimensionality, methods such as this may become useful, but will require

refinement of the operators used and the methods for controlling the projection of the data.

Bishop and Tipping [33] describe a recent, hierarchical, interactive approach to multivariate data visualization. Two- dimensional projections of the data are provided, and a statistical (hierarchical mixture of latent variables) model is used to provide the user with projections at successively increased levels of detail based on the interactive selection of regions. The method is demonstrated on a 12-dimensional synthetic data set and a 36-dimensional real data set.

A problem that resembles an inverse of those considered in this chapter is the use of neural networks as a tool for the visualization of other multivariate data. The work on Principal Component networks [65] is an obvious example, where a neural network implements PCA in an on-line manner. An MLP can also be used to perform non-linear PCA, if at least 3 hidden layers are used [30, 32]. Unsupervised neural networks, such as Self-Organizing Maps and Hopfield networks, have also been used extensively to visualize multivariate data (see [194] and the references therein).

3.4 Visualization in the Neural Network Research Domain

The area of visualization (in its own right) in neural networks is one which, surprisingly, has received little attention ([56] being the only review article to the author's knowledge). While a number of visualization methods have been developed in the process of pursuing other goals, it would seem that there is still much that can be gained through the development of special visualization methods for neural networks and other complex systems involving high dimensional data.

3.4.1 Issues in Neural Network Visualization

In this chapter a distinction is made between two general goals which exist in the area of visualization and feed-forward neural networks. The main subject of interest may be to address the question of *understanding or interpreting the mapping produced by a trained network.* This has been a long-standing interest in neural network research, because of the fundamental "black-box" property of neural networks. While it may be relatively easy to apply a neural network to some given problem, it is another matter to attempt to understand the solution which the network produces. This situation is in contrast to other classification methods such as decision trees,

where the model is directly interpretable as a set of Boolean rules. For examples of research in this area which are concerned with visualization issues, see [128, 165, 177, 196, 199, 227, 256]. This work is also invariably related to the area of *Rule Extraction* from neural networks (see, e.g. [3] for an introduction and references). This active area of work is however outside the scope of this thesis and is not discussed further.

The second subject, which is the focus of this thesis, is to attempt to gain a more fundamental insight into the network itself using visualization and exploratory techniques. This work focuses primarily on the dynamics of learning algorithms and the structure of the error surface and its relation to the training set and to the fixed properties of the network configuration. There are a number of other quantities that can be examined, such as the input, hidden and output unit activation spaces. Some work in this area is discussed below.

The hope of taking such an approach is that some useful practical insights can be gained into the general mapping properties of MLP's, and be able to more fully understand and compare the behaviour of different training algorithms and procedures.

MLP's are used to implement mappings from input space to output space by adjustment of their weights. As such, the information encoded by an MLP, and the representations formed in response to training, are primarily described by the values of the weights. Weight space is of very high dimensionality for all but the simplest of network architectures, and the dimensionality scales roughly as $O(n^2)$ in the number of nodes in the network. Hence, the problem involves the understanding of high-dimensional, multivariate data and visualization is now examined as a technique for exploring such data.

3.4.2 Visualization Techniques Developed in Neural Network Research

In this section some of the techniques that have been developed within the neural network research community for visualization are surveyed. These techniques are in addition to methods discussed in preceding sections, where previously existing visualization techniques have been applied to neural networks.

Hinton and Bond Diagrams

The Hinton diagram [97] is one of the most well-known methods used in the visualization of neural networks. In a Hinton diagram (Figure 3.8), each weight value in the network is

represented as a box. The size of the box gives the magnitude of the weight, whereas the colour (e.g. white or black) of the box indicates a positive or negative weight respectively. The connectivity of each unit is plotted separately in a larger box, which separates input and output weights to the unit. "Unit boxes" are plotted in a Draftsman's display (which may be further organized to reflect the layers of the network, if space permits). Bias weights are distinguished by introducing a third layer of weight boxes into each unit box [216], although this could also be done through the use of more colours, or the introduction of a second shape or line style.

Hinton diagrams are an application of the representation of data through features of objects (glyphs) as discussed above. Although this provides a useful preliminary visualization, the diagram only conveys information concerning the magnitude of weights in the network, and so reveals little about how a network behaves during training [249]. In practice, Hinton-style diagrams have been used to examine the mappings produced by trained networks, and further to experimentally adjust the number of hidden units to optimize performance [176] (see also [137]).

The Bond diagram [248, 249] visualizes the weights on the topology of the network. Units are represented as simple points, with "bonds" of varying length (weight magnitude) and colour (weight sign) emanating from unit outputs towards other units (Figure 3.9). Bias weights can here be represented as coming from bias units, or can be distinguished by representing each unit as a circle of radius proportional to bias magnitude and colour proportional to the sign of the bias [216]. It is argued that the bond diagram is more readily interpreted because the network topology is more prominent, and that it is better suited to an animated display [249]. While the authors acknowledge that the bond diagram may not scale up effectively to large networks, they argue that in this case the visualization of portions of networks may be worthwhile.

Trajectory Diagrams

Wejchert and Tesauro also consider a trajectory diagram (Figure 3.10), which emphasizes the visualization of the learning process itself [248, 249]. They suggest representing the multidimensional coordinate system in the 2-D plane by projection. This makes the axes non-orthogonal and subsequently the points plotted non-unique. Nevertheless, the "star-like" projection allows weight vectors to be plotted component-by-component radially via a simple transformation onto the 2-D plane. This kind of visualization can be viewed as the network is trained, leading to

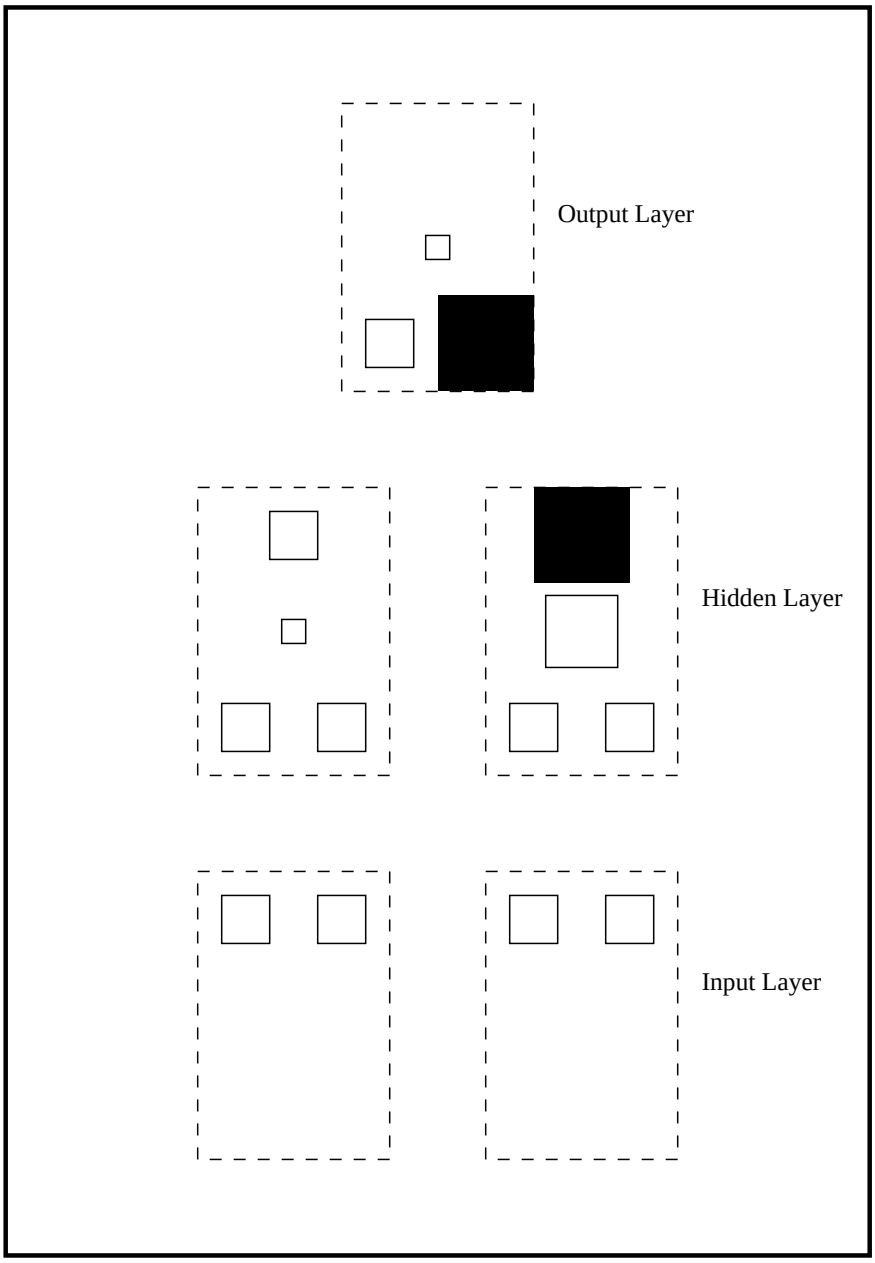

Figure 3.8: Hinton diagram for a network that solves the XOR problem.

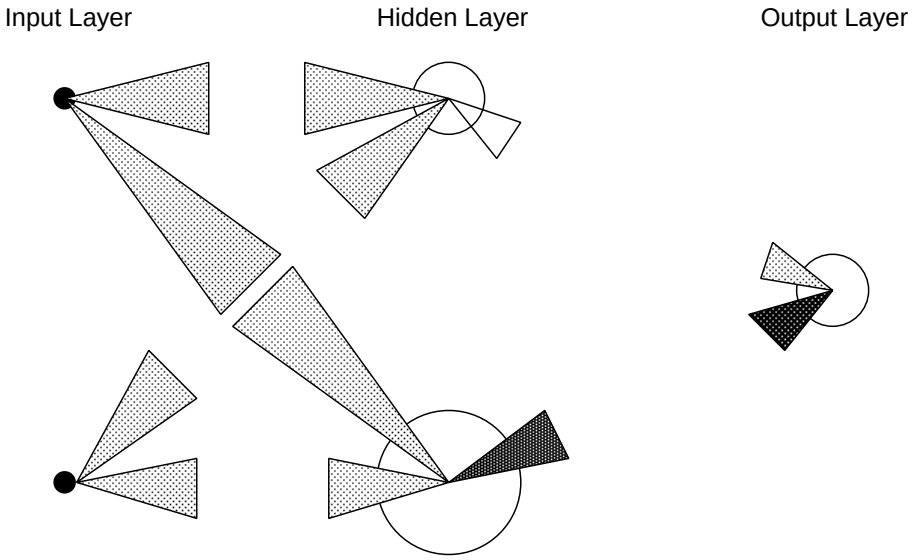

Figure 3.9: Bond diagram for a 2-2-1 MLP network.

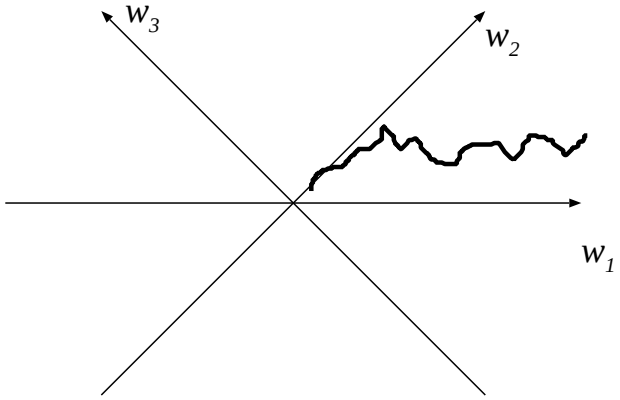

Figure 3.10: Illustration of a trajectory diagram.

a trajectory of points through weight space which is followed by the learning algorithm. It is claimed that this gives insight into the nature of the error surface in the region of the trajectory; examples are presented on up to six-dimensional groups of weights (such as those input to a hidden unit). Although useful, the technique seems limited for the very high-dimensional data produced when many practical size networks are used. A visualization method using PCA is proposed below, which can be viewed as a modification of the trajectory diagram.

Slices of the Error Surface

The most common attempt to visualize the error surface in the literature is to plot ranges of two different weight values against the error function, producing a surface which is actually a two dimensional "slice" of the true N-dimensional error surface. Such an approach is intuitively

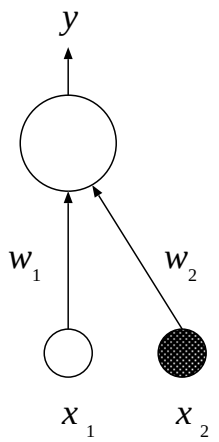

Figure 3.11: A simple network with only two weights.

appealing, and has been used extensively in the literature [32, 96, 111, 110, 120, 154, 166, 175, 215]. Hush and Horne in particular use this method of visualization to describe and illustrate the general properties of error surfaces [111, 110]. This is discussed further in Chapter 4.

The only error surface that can be completely represented in this way corresponds to a network with only two variable weights, such as a single unit with one (variable) input and one bias weight ($x_2 = -1$). This network is shown in Figure 3.11.

Although this network is of little practical importance, some insight can be gained from visualizing the complete error surface of this net. Note that the network computes a mapping $y = 1/(1 + e^{-(x_1 w_1 + w_2)})$; $x_i \in I\!R, y \in (0, 1)$ with a sigmoidal activation function. To demonstrate this visualization, the SSE error function and the following learning scenarios are considered:

1. Two binary training pattern input-output pairs: $\{(0, 1), (1, 0)\}$.

2. A single randomly generated real-valued training pattern pair.

3. A training set of 10 randomly generated patterns.

All randomly generated input and desired output training values were chosen in the interval $x_i, y_i \in (0, 1)$. The error surfaces for these problems are shown in Figures 3.12- 3.14.

The first task is to invert a single bit; the output being a 1 for a 0 input value and a 0 for a 1 input value. The shape of the error surface can be easily understood by considering the sigmoid as approximating a threshold function. Each pattern can then be written as an equation of the

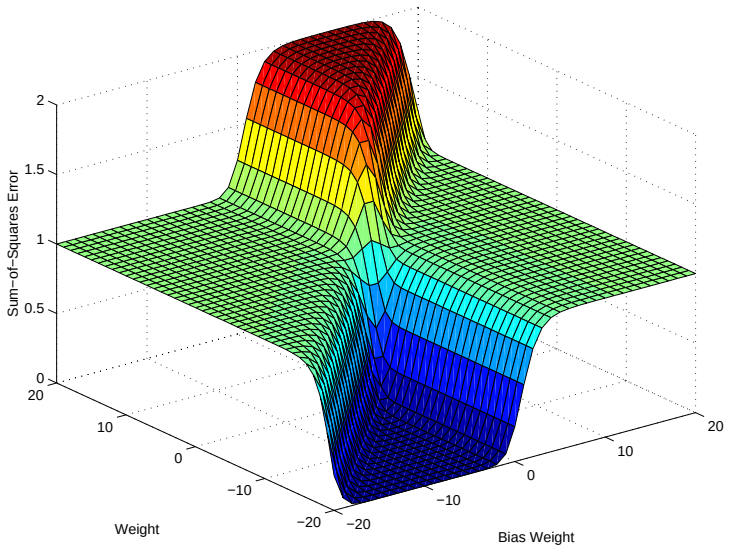

Figure 3.12: The error surface for a network with two weights, using two binary training patterns.

form

$$z = xw_1 - w_2$$

which, when the input and desired output values are substituted, describes the line on which the threshold transition occurs. For these patterns, the equations are

$$\begin{aligned} 1 &= -w_2 \\ 0 &= w_1 - w_2 \end{aligned}$$

The equations indicate that the network will produce the correct output for the first pattern (0,1) when $w_2 > -1$, the correct output for the second pattern (1,0) when $w_1 > w_2$, the correct output for both patterns when both of these inequalities are satisfied, and the incorrect output for both patterns when both inequalities are violated. These conditions lead to three possible different error values on the error surface. In Figure 3.12, the error surface is shown when a sigmoid is used as the transfer function, which leads to a "softening" of these threshold boundaries between different pattern classifications. The plateaus of this error surface extend asymptotically towards infinite values of w_1 and w_2, as the sigmoid approaches its asymptotic values.

In the second task the real-valued pattern requires w_1 and w_2 to take on values such that the sigmoid input is equal to the inverse sigmoid function of the desired output value. In Figure 3.13

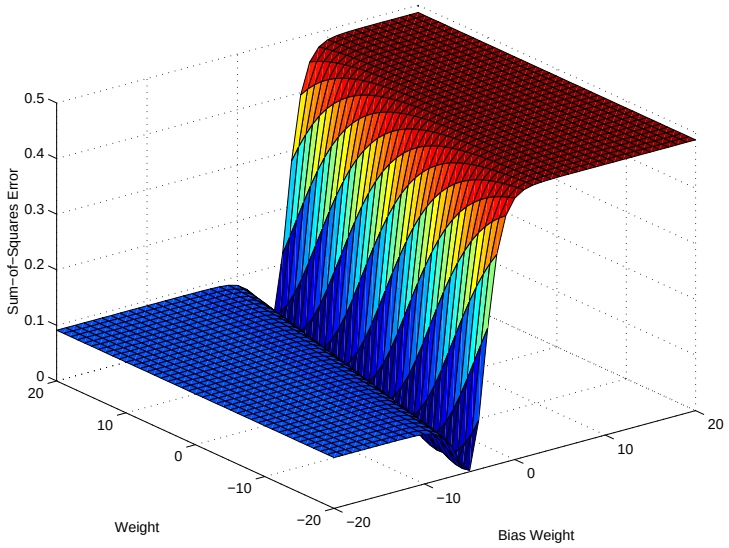

Figure 3.13: The error surface for a network with two weights, using a single random real-valued training pattern.

this is satisfied by a line on the error surface. This demonstrates how the ravine-like structure and asymptotics of the error surface can be produced quite simply by the properties of the network mapping. Also note that the binary patterns used in the first example can influence the appearance of plateau or staircase-like features.

Figure 3.14 shows the effect of increasing the size of a training set of randomly generated patterns. It is clear that each pattern contributes features to the error surface, since the patterns are combined through the error function. These contributions lead to the complex structure shown. An increasing number of patterns can also produce areas of smoother transition on the error surface, due to this combination in the error function.

Visualizing the error surface in the above way has the potential to provide a great amount of information to the user. Humans are extremely good at perceiving a three-dimensional (i.e. two weight dimensions plus height/error) surface on a 2-D computer display or sheet of paper, based on a wealth of life experience. In addition, people can often manipulate such images cognitively and “on the fly”, performing transformations and extrapolating over hidden regions. It is easy to extract features of a surface and think about the issues involved in searching it for a minimum point. Furthermore, algorithms can be applied to the task and their trajectories can be plotted onto the surface, perhaps dynamically as training progresses.

Despite these informative results, there are a number of serious difficulties in using 2-D slices of an error surface of higher dimensionality to infer properties of its structure. The prob-

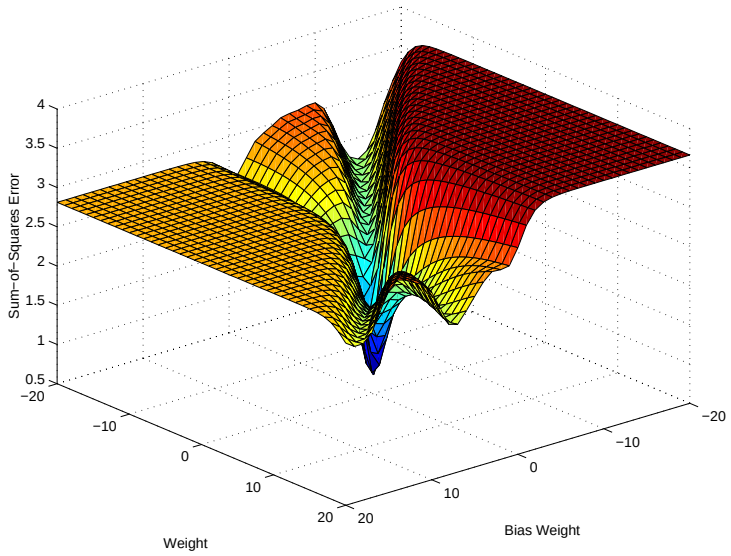

Figure 3.14: The error surface for a network with two weights, using a set of 10 random real-valued training patterns.

lems stem from the very high loss of information which results when all but two dimensions of a very much higher dimensional space are ignored. It is unclear whether such a visualization provides any useful information in terms of the error surface as a whole or the path followed by a learning algorithm through weight space. The error surface can be surprisingly complex, even for simple problems such as XOR, so that visualizing 2-D slices of it may provide misleading information [87].

Androulakis et al. [4] also examine the structure of the error surface using 2-D projections. An initial point on the error surface is chosen, and the maximum and minimum eigenvalues and corresponding eigenvectors of the Hessian are found at this point. The two dimensional space spanned by these two eigenvectors is then explored in a grid-like manner, by changing the initialization point to each position on the grid, and running the training algorithm from this point. Each position on the grid is colour coded according to:

- Whether the algorithm converges or not (given some prior convergence criteria) - non-convergence is shown as white.
- The location of convergence - different minima are assigned different base colours.
- The rate of convergence - dark colour shades are used for faster convergence, lighter shades for slower convergence.

The technique is used to compare the performance of several different training algorithms

for the XOR and a 1-D sine function approximation problem. The authors claim that the method is useful for comparing the performance of different algorithms, as well as the sensitivity of the relationship between an algorithm and the error surface. However, usage of the method still requires the initial point (for which the eigenvectors are calculated) to be specified in advance, which is difficult to do when the location of minima or other special points are unknown. Also, the 2-D subspace chosen is still effectively a dimensionality reduction, and so may not reveal important information lost in the projection operation.

A draftsman's display of 2-D slices of the error surface is usually not feasible because of the large number of weights in many practical networks. Lawrence [137] uses suggests a draftsman-like method of plotting many 2-D slices of the error surface, choosing two weights at random from the network for each plot. This seems to provide a better global picture, but introduces the problems of a draftsman's display mentioned above. The interpretability of a single surface is rapidly reduced with the size of such a display.

Hyperplanes, Hidden-unit Representations and Interpreting network functions

A number of researchers have examined different data spaces in MLP's other than weight space. Perhaps the most widely-used concept is that of hyperplanes in the input space [162]. Consider a linear threshold unit which implements a mapping from its two-dimensional input space to $0, 1$. The output of the unit is:

$$y = sgn(w_1 x_1 + w_2 x_2)$$

that is, the output of the unit is 1 if

$$(w_1 x_1 + w_2 x_2) > 0$$

otherwise it is zero. This inequality represents the "decision" of the unit, and can be visualized as a line in the two-dimensional input plane. This concept extends naturally to higher-dimensional spaces, where the decision threshold is a $(N-1)$-dimensional *hyperplane* in an N-dimensional input space. Figure 3.15 shows the (1-D) hyperplane which correctly classifies the 2-bit AND function. Note the arrow attached to the hyperplane indicates which side of the hyperplane corresponds to an output of 1.

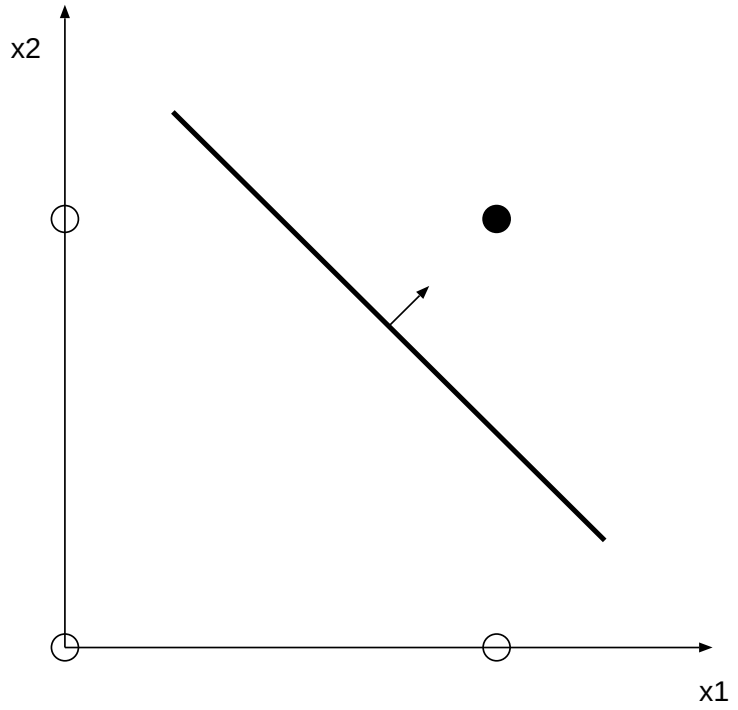

Figure 3.15: A single hyperplane (which can be implemented using a single threshold unit), solves the Boolean AND function.

Next, consider a 2-2-1 MLP with threshold activation functions. Each hidden unit can be considered in turn as implementing a single hyperplane in the 2-D input plane. These decision regions are combined by the output unit to form the decision regions of the final network. Figure 3.16 shows an idealization of how such a network can solve the XOR problem, by using a combination of two hyperplanes in the input space.

Visualization of hyperplane-constructed decision regions has been used in the literature to examine how a network solves a given problem [96]. An extension can be made by viewing a sigmoidal function as a smoothed approximation to a threshold function, and plotting the input space shaded with varying levels of grey to represent output values between 0 (black) and 1 (white). Such displays are sometimes called *response function* plots [56]. The so-called "Two-Spirals" problem [136] is a problem to which this visualization technique is often applied. The task is to classify points generated from two inter-twined 2-D spirals according to the spiral they belong to. The problem requires a highly non-linear solution, and the decision regions constructed by MLP's are often highly complex and varied.

It is also possible to use hyperplanes and response function plots to visualize the "decision regions" of the hidden layer "activation" or output space [158]. In this way it can be seen how a hidden layer can transform a given problem into a linearly separable one, which is then

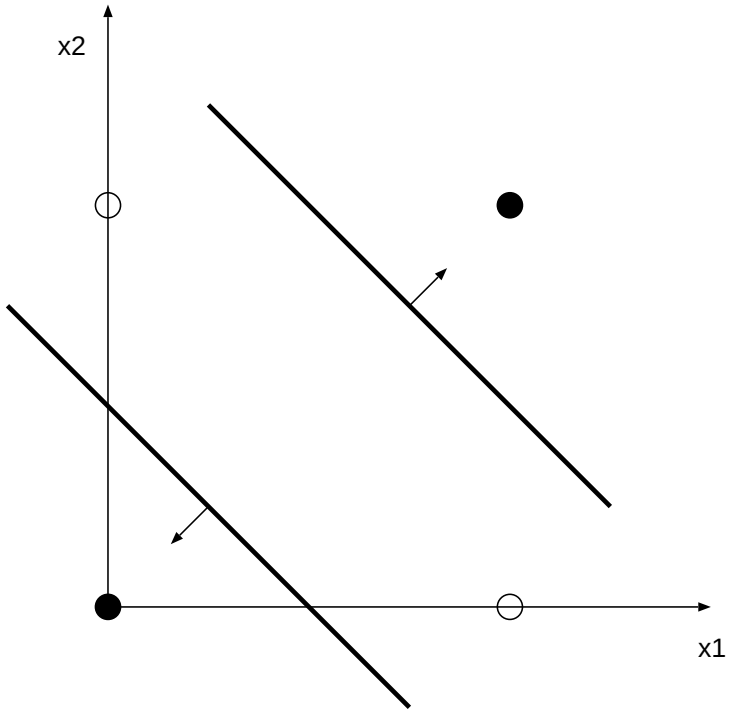

Figure 3.16: A solution of the Boolean XOR problem using two hyperplanes.

solved by the output layer of weights. This method has been used in the analysis of *N-2-N* encoder networks [12, 13, 143, 144, 158] (see also Section 4.1.3). It is shown that a certain symmetric arrangement of the hyperplanes is required to solve the encoder problems, which becomes more tightly constrained as the number of inputs/outputs N is increased. This insight is used to explain the difficulties observed with training encoder networks with backpropagation and other algorithms [12, 144].

The main limitation with hyperplane diagrams and response function plots is that it is only possible to visualize a two or perhaps three dimensional space in this way. Although this has proven useful in some situations, such as for problems like two-spirals and for autoencoder networks, most situations involve spaces of higher dimensionality.

3.4.3 Software Packages and Simulators

ANN's have enjoyed a significant amount of interest in the past decade or so, both within the research community and in application to an extremely wide range of areas. This interest has led to the development of a number of software simulation and analysis packages (for a list, see [51]). In this section some of the work in this area is discussed which is at least partly concerned with visualization issues.

Neural networks are conveniently implemented within a spreadsheet package [29]. In addition to being relatively straightforward, spreadsheets typically already have the ability to produce simple visualizations of various quantities. In [29] Bergeron uses these capabilities to produce bar charts of input, hidden and output unit activation and weight values input to a given node. It is possible to view these and other quantities dynamically (e.g. change in output values for each input pattern as training progresses).

A number of software packages produce various one, two and 3-D displays of training and related MLP data. For example, the Neurograph [254] package uses a plot of error as a function of training epochs graph, a Hinton diagram, a bond-diagram-like display and bar charts showing the error resulting from each pattern in the training set. Jones et al. [119] describe a visualization system which can be applied to neural network data, to produce 3-dimensional plots of (2-D) slices of the error surface (as described above) and contour plots of these 2-D slices. The Neuralis software [127] plots "single-point" multivariate quantities such as activation values for input patterns. The output of each output neuron is also displayed, together with the target values for patterns during training (see also [85]).

Liao and Moody [141] describe a recent software toolkit in which the focus is on the visual interpretation of a previously trained network and the corresponding training data set. The sensitivity of the network inputs and hidden unit are examined by measuring the hidden unit activations for each pattern. Desired versus actual outputs, as well as several different gradient-based measures, are displayed averaged over the entire training set or examined individually. The tools may be used to assess the importance of inputs to the task at hand interactively, to select those features for training and to prune hidden units to improve training performance.

3.4.4 Discussion

This section has considered the area of visualization applied to ANN's, in particular MLP networks. While a number of useful approaches have been developed, this is an area in which a relatively small amount of work has been done. Significant advancements in computer technology (especially graphical capabilities, processing speed and storage capacity) have created the environment to support the development of new methods of visualizing the high-dimensional data associated with MLP's.

Visualizing MLP weight data is quite a novel problem in that this weight data is typically of very high dimensionality. Most current methods proposed in this area scale awkwardly to networks with more than a few weights. In the next section, a method of visualization is developed using PCA, which is more robust in this respect and remains flexible enough to be used in a general MLP training environment.

3.5 Principal Component Analysis

Principal Component Analysis (PCA) is a technique widely used in the statistical community, primarily for descriptive but also for inferential purposes [112, 117]. PCA is also known as the Karhunen-Loeve transformation (KLT) or expansion in the signal processing literature.

Let $\mathbf{w} = (w_1, \ldots, w_N)$ be an N-dimensional weight vector containing all the weights in a network. This vector is defined by a single point in weight space (or equivalently on the error surface). The weight space coordinate system defines each $\mathbf{w}$ as a linear combination of N orthonormal basis vectors $\mathbf{u}_i$

$$\mathbf{w} = \sum_{i=1}^{N} w_i \mathbf{u}_i .$$

Training the network for s epochs produces a trajectory $\mathbf{R}$ of the learning process moving on the error surface

$$\mathbf{R} = \{\mathbf{w}^j\}, \;\; 1 \leq j \leq s.$$

The training process produces a set of data points $\mathbf{R}$ in weight space, on which PCA can be performed. An efficient method of doing this is to firstly calculate the covariance matrix Σ of the trajectory $\mathbf{R}$

$$\Sigma = \frac{1}{s} \sum_{j=1}^{s} (\mathbf{w}^s - \overline{\mathbf{w}})(\mathbf{w}^s - \overline{\mathbf{w}})^T$$

where $\overline{\mathbf{w}}$ is the sample mean vector

$$\overline{\mathbf{w}} = \frac{1}{s} \sum_{j=1}^{s} \mathbf{w}^j$$

and then calculate the eigenvectors and eigenvalues of the covariance matrix

$$\Sigma \mathbf{v}_i = \lambda_i \mathbf{v}_i .$$

The eigenvectors provide a new set of basis vectors and coordinate system

$$\mathbf{w} = \sum_{i=1}^{N} b_i \mathbf{v}_i,$$

mapping vectors from $(w_1, \ldots, w_N)$ to the new coordinate system $(b_1, \ldots, b_N)$ (ie, a rotation of weight space). To reduce the dimensionality of the weight space to some value $d < N$, the d largest eigenvalues and corresponding eigenvectors are chosen and the remaining $N - d$ values discarded. Projecting $\mathbf{T}$ onto this d-dimensional subspace means replacing $N - d$ of the coefficients z_i by constants c_i so that each vector $\mathbf{w}$ is approximated as

$$\tilde{\mathbf{w}} = \sum_{i=1}^{n-d} b_i \mathbf{v}_i + \sum_{i=(n-d)+1}^{n} c_i \mathbf{v}_i.$$

Performing this dimensionality reduction results in the smallest possible loss of (variance) information, with regards to the sum of squares error function, for any other similarly defined choice of basis vectors. This minimizes the L_2-norm between the original distances between points and their projected distances. The direction of greatest variance[1] in the data is given by the first principal component (PC) $\mathbf{v}_i$, corresponding to the largest principal value (PV) λ_i.

Note that PCA can also be conducted using the Singular Value Decomposition (SVD) result from matrix theory. SVD provides a more computationally efficient method of performing PCA, and has found other applications in the analysis of the PCA procedure [117].

In practice, PCA is quite often performed on the correlation matrix [117] rather than the covariance matrix. The main reason for this is that if the variables in question are measured in different units, it may be that the variances of some variables dominate the first few principal components, simply because of the units they are measured in (e.g. consider one variable measured in meters together with some other variables measured in millimeters). In the case of neural network weight data, this problem does not arise - the covariance matrix is therefore used for PCA in the following sections.

It is also worth mentioning that PCA and neural networks have been brought together in a different situation. Consider an autoencoder-MLP with no bias unit and N_h linear hidden layer units. It has been shown that if such a network is trained with a least squares error criterion, then

[1] This is sometimes referred to as the largest energy component of the projection

the best possible fit occurs when the hidden units span the subspace of the first N_h PC's [14, 194]. Much work has developed in this area, with one major motivation being the exploration of biologically plausible neural network architectures and learning schemes which perform PCA iteratively in such a way (for more details, see [65]).

3.6 Weight Space Learning Trajectory Visualization

PCA is often used to create a projection of multivariate data onto a space of lower dimensionality, for visualization purposes, whilst attempting to preserve as much of the structural nature of the data as possible [117]. If the data can be well represented in a two or three-dimensional subspace (which is captured by the first two or three principal components), it is very convenient to directly plot this projection on a computer screen. Further, since many of the techniques discussed in Section 3.3 have proven effective for higher-dimensional data (usually $N < 10$), if the data can be effectively reduced to this kind of dimensionality, these techniques could be used following PCA.

3.6.1 Experimental Results: Student-Teacher

In this set of experiments, an artificial learning task was used, which is sometimes referred to as the *student-teacher* learning model [164]. In this problem, two networks are created. One network, the teacher, is initialized in some way, and then represents the "oracle" for the learning task. The training set is produced by generating random input vectors and passing these inputs to the teacher network, producing the desired output vectors. The other network, the student, is then trained using this data.

Three MLP network configurations were chosen; 2-2-1, 4-4-1 and 8-8-1, which correspond to N = 9, 25 and 81 weights respectively (including bias weights). For each configuration, 10 problems were generated and used. In addition, 10 separate training runs were conducted for each problem, commencing from a different (randomly generated) weight initialization (range [-0.25,0.25]). The random problems were produced by creating a teacher network of identical configuration to the learner, with weights generated from an $\mathcal{N}(0, 1)$ distribution. Thus a total of 300 training sessions were conducted. Standard back-propagation was used, with learning rate $\eta = 0.1$. A training set of 10^4 input patterns was randomly generated for each of the 10 problems

Network	PC1	PC2	PC3	PC4	PC5
2-2-1	93.57 (6.16)	5.08 (5.28)	1.01 (0.94)	0.21 (0.15)	0.08 (0.07)
4-4-1	89.46 (7.18)	7.21 (5.48)	1.68 (1.32)	0.53 (0.41)	0.31 (0.27)
8-8-1	80.59 (7.62)	8.47 (3.46)	3.05 (1.37)	1.64 (0.77)	1.09 (0.49)

Table 3.2: Percentage of variance captured by PCA.

(each element of each input pattern being drawn independently from a $\mathcal{N}(0,1)$ distribution), and the required outputs for each pattern were produced using the teacher networks. Weights were updated in stochastic mode (i.e. after each pattern presentation), using examples chosen at random from the training set. Note that this "student-teacher" model of learning guarantees the presence of global minima (ie, points which are functionally equivalent to the teacher weight configuration), *where the value of the error function is zero*[2]. Each network was trained for a total of 10^4 epochs, and the values of the weights were recorded after every tenth epoch.

A summary of the results is given in Table 3.2. Shown are the network configurations used, and the average percentage of variance captured by each of the first five principal components (each value being averaged over the 10 problems and 10 separate training runs for each network topology). Shown in brackets is the standard deviation for each mean value. Clearly, the majority of the variance in the weight data is captured by the first principal component. The results reveal a decreasing trend in the average first principal value as the number of weights increases. This trend is to be expected, as increasing the number of weights increases the number of possible dimensions in which the training process can move. The variance "lost" by the first principal component as the number of weights increases is picked up by the remaining principal components but remains heavily skewed towards the first values. What these results reveal is that whilst training is a search across an N-dimensional error surface, the *intrinsic dimensionality* of the path taken by back-propagation is in fact much smaller than N. The learning path along the first few principal components (in fact, usually only the first one or two), can therefore be viewed without any significant loss of information.

An example of the visualization provided by the method described above is shown in Figure 3.17, which shows the mean-squared error (MSE) plotted against the first two principal components for a 2-2-1 network configuration. For this particular example, the first two principal components capture 83.00% and 14.12% of the variation in the data respectively. Plotting

[2]The teacher weight vector and the $2_h^N N_h!$ symmetrically equivalent weight vectors (see Section 4.1.4) correspond to global minima.

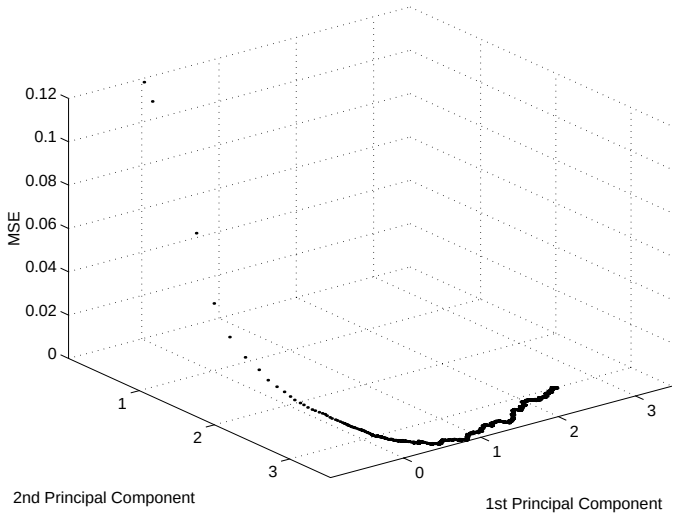

Figure 3.17: Example of a Learning Trajectory approaching a global minimum.

point values in this way gives some idea of the velocity of the trajectory (ie, rapid drop in MSE and location initially, progressing to small movement towards the end of learning). Initially, the trajectory moves in a direction close to that of the second principal component, but a region of low error (MSE ≈ 0.0003) is rapidly found where the network remains for the rest of training, moving in a direction similar to that of the first principal component. Clearly, the trajectory has revealed that from this particular starting point, there is a relatively steep gradient on the error surface (in the direction of the second principal component), which leads to a region of low error, sloping gradually in the direction of the first principal component. This final region must also be somewhat flat, as there is only a small amount of movement in the direction of the second principal component, and almost no movement ($< 2.88\%$ of total variance) in any other direction. The fact that back-propagation slows down as the error becomes small is also observable, from the amount of time spent following the gradient along the first principal component.

In Figure 3.18, a 4-4-1 configuration network has been trained, and the training process has become trapped in a region of attraction; MSE ≈ 0.0155. The trajectory moves steadily in the positive direction along the first principal component. For the second principal component the direction of movement gradually changes from positive to negative. The overall displacement in the direction of the second principal component is quite small, implying that movement along it has contributed little to the change in MSE. Similar behaviour is observed for higher principal components, indicating that the small amount of information lost in considering only the first

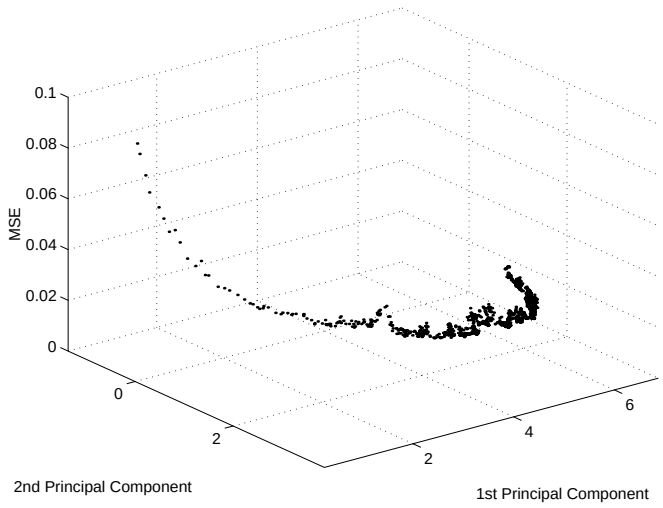

Figure 3.18: Example of a Learning Trajectory approaching a local optimum or flat plateau of the error surface.

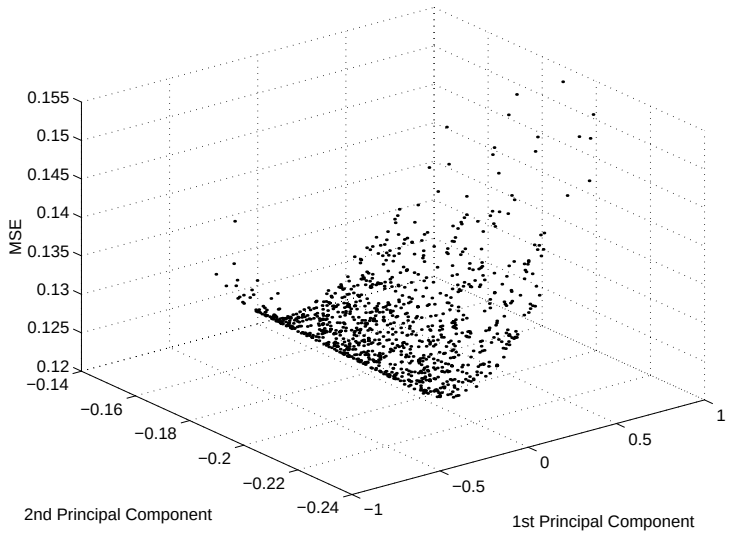

Figure 3.19: Local minimum learning trajectory behaviour for one example problem. The trajectory moves from the top-left to bottom-right of the figure. Note the different scales on the axes.

one or two principal components is not significant to the overall visualization of the learning process. For this training run, 80.69% of variance was captured by the first principal component, and 9.33% by the second. In comparison to Figure 3.17, this trajectory is noticeably "rougher"; the underlying direction followed being affected by random noise-like effects in the error and direction. This shows that the error surface is subject to noticeable changes in error over small changes in its position. Unlike the example shown in Figure 3.17, a smooth path was not found in following the gradient of this error surface from this starting point. The other training runs conducted on this problem revealed similar behavior, suggesting this is a general property of the error surface (and therefore highly problem dependent). The majority of the curves produced for the 8-8-1 networks revealed similar behaviour to that shown in Figures 3.17 and 3.18.

While most of the results produced over all experiments were fairly "well-behaved" in that the trajectory dropped quickly to some minimum and remained there (as in Figure 3.17 and 3.18), this is not always the case. Figure 3.19 shows the trajectory produced by one of the 2-2-1 networks. This kind of trajectory was observed for 9 of the 10 training runs conducted on this particular problem. Clearly, the choice of starting point initialization for this problem often places the network in a sub-optimal region of attraction, where it remains throughout training. The fact that weight updates cause oscillatory behavior suggests that the network is trapped in a region with steep gradient in both directions along the first principal component. For this particular problem the first principal component captures 99.52% of the variance, indicating that the network is trapped in a steep "rain-gutter-like" local region, with steep positive gradient along the first principal component and almost no gradient in all other directions. The trajectory begins at the top of Figure 3.19 and moves slowly in the negative direction of the second PC.

3.6.2 Experimental Results: Real-world Datasets

Further experiments were conducted on networks in a realistic learning situation. The datasets used here are part of the benchmark problem collection "Proben1" [181]. The first problem is the "Credit screening" dataset, which consists of 51 real inputs and two Boolean outputs. The task is credit card approval on the basis of the inputs. A 51-15-2 network was used for this problem (a total of 812 weights, including bias weights) and the credit card approval dataset contained 690 patterns. The second problem is the "Wisconsin breast cancer" database, which consists of 9 real inputs, and two Boolean output variables. The outputs represent a classifica-

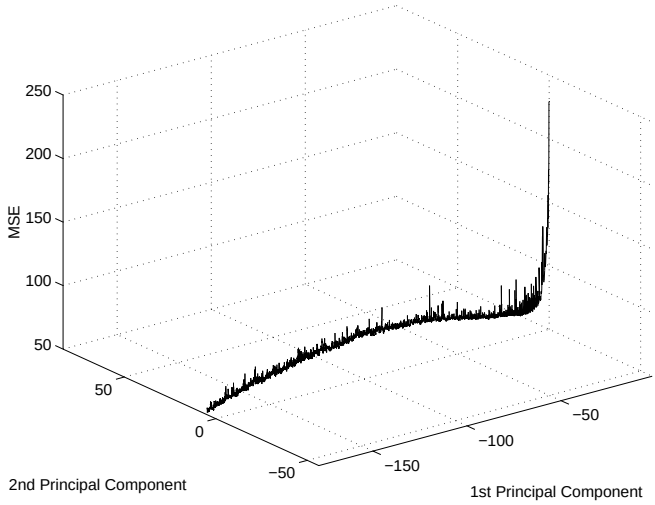

Figure 3.20: A learning trajectory visualization example for the credit card training problem.

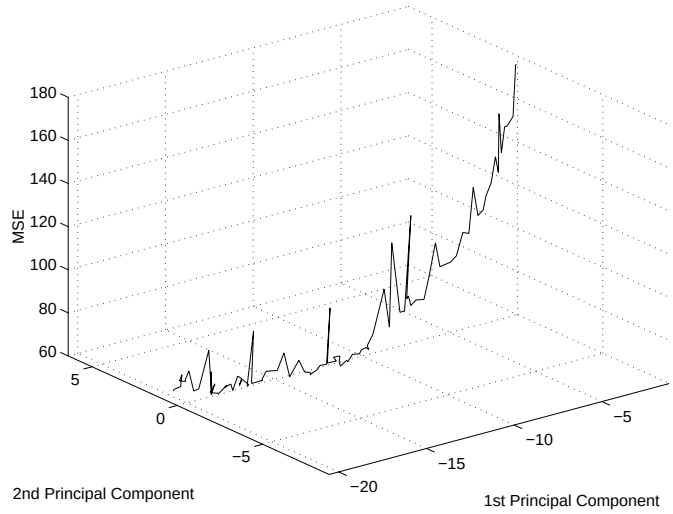

Figure 3.21: A closer view of the first 100 weight recordings for the example shown in Figure 3.20. Note the difference in scales used.

tion of benign or malignant on the basis of the inputs. For this problem a 9-5-2 network was chosen (62 weights). The entire dataset of 699 examples was used to train the network. For both problems, the network was trained 10 times from a random initialization, using standard backpropagation with learning rate $\eta = 0.25$. Backpropagation with stochastic weight updating was used. The networks were all trained for 10^5 pattern presentations, and the weights were recorded after every 20 pattern presentations.

An example of the visualization provided by PCA for a real world problem is shown in Figure 3.20. This Figure shows the learning trajectory produced by training one of the networks learning the credit card approval problem. Here the cost function (MSE) is plotted against the

Dataset	PC1	PC2	PC3
Card	96.57 (0.78)	2.34 (0.49)	0.56 (0.20)
Cancer	89.74 (2.17)	7.28 (1.23)	2.31 (1.24)

Table 3.3: Percentage of variance captured by the first three PC's.

first two PC's. The first PC accounts for approximately 97.50% of the variation in the trajectory, while the second PC accounts for a further 1.83%. As with the student-teacher experiments, *this learning trajectory can be represented in a very low dimensional subspace to very high degree of accuracy*, (i.e. with a minimal loss of information). In fact, the trajectory moves almost entirely along a straight line in weight space, which is defined by the first PC. This visualization reveals a rapid reduction in the value of the cost function initially (corresponding to an area of steep gradient), which then changes to much slower progress for the remainder of training (consistent with a "plateau-like" area, with very shallow gradient). The curvature observable in the trajectory is not an increase in error but some slow oscillating movement in the direction of the second PC.

Figure 3.21 is produced from the same training run, but in this case a particular portion of the trajectory, the first 100 weight recordings, is examined, and PCA conducted only on these data. This visualization allows a closer view of a portion of the trajectory, however the resulting PC's will be different from those obtained from the entire trajectory, because only a subset of the data for the complete trajectory is being used. In Figure 3.21, 94.58% of the variance is captured by the first PC, and 2.20% by the second. The fact that the weights have been updated stochastically has introduced an element of "noise-like" behaviour (allowing temporary increases in the cost function over the whole dataset), which can be more clearly seen here in comparison to Figure 3.20.

Figure 3.22 shows a learning trajectory for one of the cancer dataset experiments. In this case, the first and second PC's capture 91.58% and 6.89% respectively of the variance of the trajectory. The trajectory moves largely in the positive direction of the first PC, but movement in the direction of the second PC changes from positive to negative, producing the curvature of the trajectory shown. Note that if required, successive 2-D plots can easily be produced for any PC against MSE, so that the value of the technique is not conditional on such a high degree of variance being captured by the first PC. Figure 3.23 shows the movement of the cancer experiment trajectory of Figure 3.22, in the directions of the third and fourth PC, which account

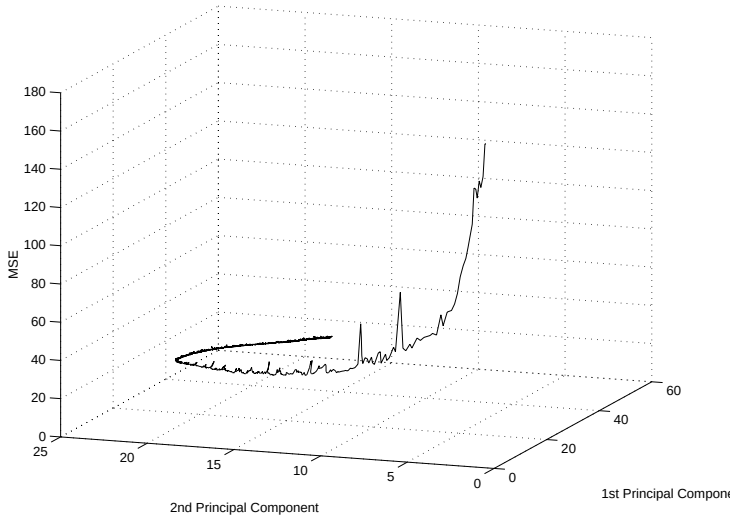

Figure 3.22: A learning trajectory visualization example for the cancer training problem.

for 0.89% and 0.40% of the total variance respectively.

For both the cancer and credit-data experiments, the degree of variance captured by the first PC's was very high overall. Table 3.3 shows the average percentage of variance captured by the first three PC's, for the two different problems. Shown in brackets is the standard deviation for each mean value. The progression of all the trajectories was approximately linear, suggesting that (for these learning problems) the nature of the learning trajectory allows PCA to be used as an effective visualization tool, avoiding some of the difficulties associated with the very high dimensionality of weight space.

3.7 Temporal Principal Component Behaviour

In many instances within statistical data analysis, PCA is conducted on samples where each data point is independently drawn from some unknown distribution. Indeed, many of the algebraic properties of the technique rely on this assumption (together with the assumption that the sampling distribution is multivariate Normal). Given MLP weight space data however, this is clearly not the case - the data can be considered as a time series of observations of the weight vector evolving in discrete time steps (epochs of the learning algorithm). PCA remains a useful exploratory method for such data [117], as such properties are primarily directed towards the use of PCA for inference.

Since the given data is a learning algorithm trajectory, it is of interest to study the evolution

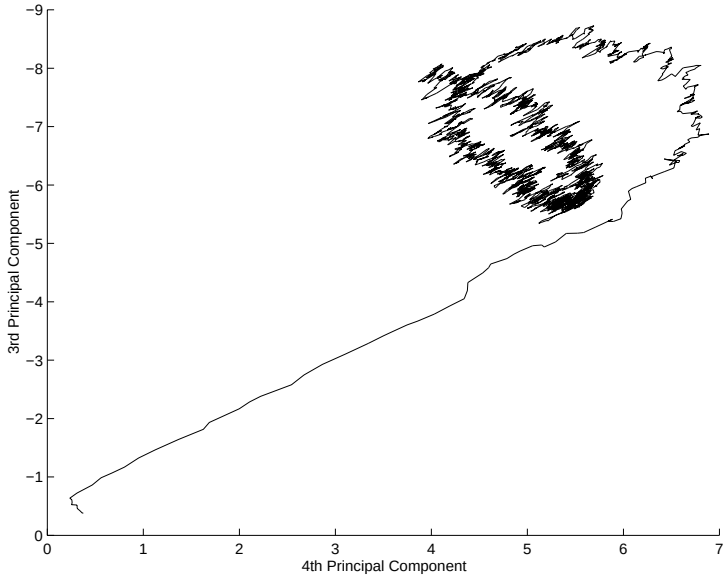

Figure 3.23: Movement of the same trajectory as Figure 3.22 in the direction of the third and fourth PC.

of the PC's and PV's as learning proceeds. The first question that might be asked is, how does the amount of variance captured by the first, second, ..., PC's change as learning progresses? This question can be answered by looking at the evolution of the PV's as learning proceeds.

3.7.1 Experimental Results

In this section the trajectories produced by training on the cancer classification problem are used to illustrate the temporal PC visualization process (results produced for the credit card problem were similar). Figure 3.24 is a plot of the percentage of variance captured by a PC as a function of pattern presentations. To produce these curves, the weight data is sampled at k equally spaced points (here, k=10 is chosen) in the data, and PCA is performed on that portion of the trajectory, $\mathbf{R}^* = \{\mathbf{w}^j\},\ j = 1, \frac{s}{k}, \frac{2s}{k}, \ldots, s$. Curves are shown for different PV's corresponding to the PC's indicated. The plot reveals how much variance the first three PC's capture as training proceeds. As can be seen, the amount of variance captured by each PC does not change to a large extent, irrespective of how much training has advanced. When the amount of variance captured by the first PC decreases, this amount is largely regained in the second PC. The curves are averages of all training runs conducted on the cancer dataset, and errorbars indicate the standard deviations involved with each point on the curves. The final points on the ends of the curves therefore show the average success of PCA in capturing the variance over

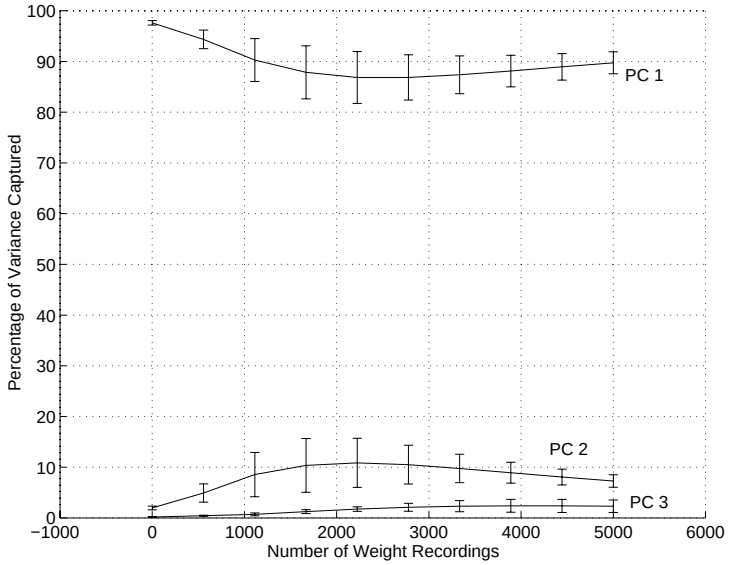

Figure 3.24: Change in the amount of variance captured by the PC's over the learning process.

all the training processes. As mentioned above, performing PCA on a subset of the complete trajectory leads to a different set of PC's, because a different set of weight data is being used.

Although Figure 3.24 indicates that the PV's (and hence the percentage of variance captured by the corresponding PC's) are similar throughout training, it is of further interest to investigate the evolution of the PC's themselves. For a given PC, the coefficients $b_1, \ldots, b_n$ indicate how "close" the direction of that PC in weight space is to each weight axis. The vectors of coefficients are the eigenvectors of the covariance matrix Σ, hence $\|(b_1, \ldots, b_n)\| = 1$. If the first PC captures most of the variance in the trajectory, a corresponding b_i of large magnitude indicates that the direction followed by the trajectory is close, in weight space, to that weight axis, and that particular weight has made a large contribution to the direction of the trajectory. The sign of the coefficients within a PC provides an obvious indication of the direction of variance of the coefficients in relation to each other (e.g. if b_i = 0.7 and b_j = -0.65), then the trajectory moves in a direction close to increasing w_i and decreasing w_j. Between PC's, the sign of the coefficients is somewhat arbitrary, as reversing the sign on all coefficients of a given PC leaves the variance captured by that PC unchanged, as well as its orthogonality with the other PC's.

Figure 3.25 shows the evolution of the coefficients of the first PC against pattern presentations for a single training run conducted on the cancer dataset. Of the 62 coefficients for this trajectory, only those coefficients whose absolute value exceeded 0.2 at one or more of the sample points are shown (which is 9 terms in this case). This visualization shows how the direction of the first PC changes as learning progresses, in terms of the weights which produced the

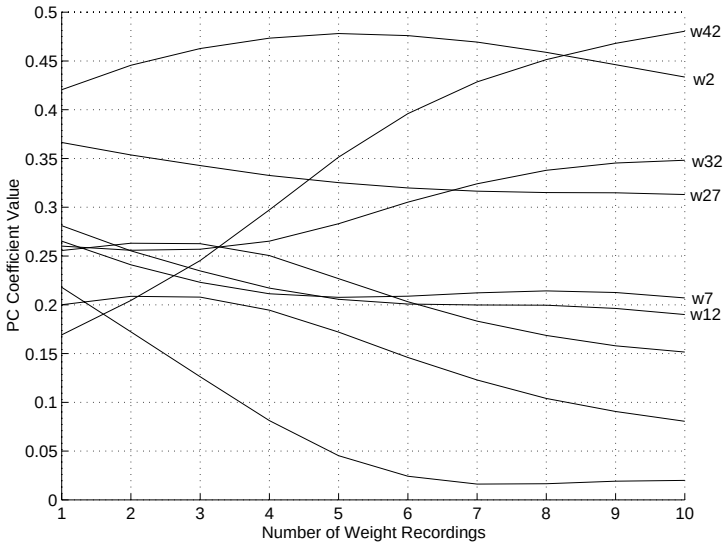

Figure 3.25: Values of the coefficients of the first PC over the learning process.

greatest variance. The fact that many of the coefficients remain small allows the determination of which of the weights contributed most to the trajectory. In examining the contributions of weights to the 10 different trajectories produced on the cancer dataset, it was found that the six weights labeled in Figure 3.25 ($w_2, w_7, w_{12}, w_{27}, w_{32}$ and w_{42}) were also among the most significant for another trajectory. Another group of six weights ($w_5, w_{10}, w_{15}, w_{30}, w_{35}$ and w_{45}) were prominent in three of the other trajectories produced (one of which is shown in Figure 3.26). If these groups are examined in relation to the structure of the network, it is found that, in each case, the six weights are *from the same inputs* to a single hidden unit in the network. Comparing Figure 3.25 and Figure 3.26 reveals a similar pattern of evolution for the six weights, suggesting that the trajectories have some symmetry, although they move in different directions. This result can be related to the known symmetry which exists in weight space due to the interchangeability of hidden units (see Section 4.1.4). On an intuitive level particular hidden units have associated themselves with particular inputs, and this happens often for different trajectories started from different initialization points. Further analysis may reveal other similar, less-obvious relationships that may provide insight into the learning process.

3.8 PCA and Neural Networks

PCA has found other applications in the analysis of MLP's, as well as in the design of enhanced training algorithms for MLP's. These applications are described in this section. Although the

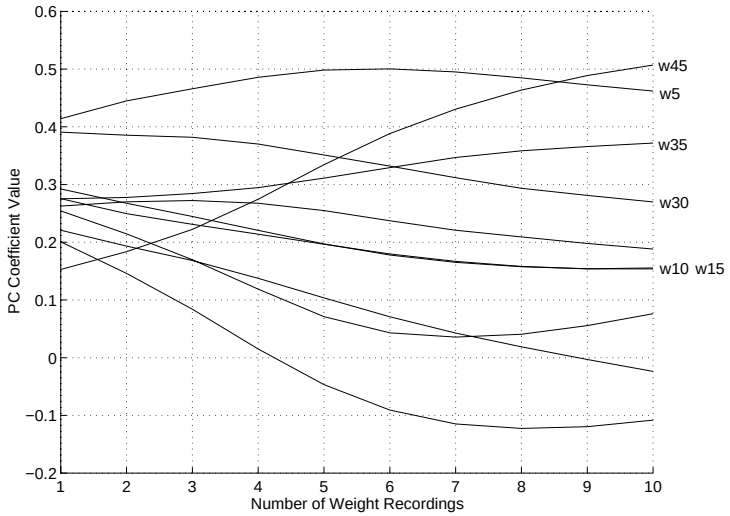

Figure 3.26: Values of the coefficients of the first PC over a different learning process.

main focus of this kind of research has not been visualization, it is obviously related to the work in this thesis. In particular, analysis of MLP's, training data and the dynamics of training algorithms deal with many of the issues involved in providing useful visualization methods. A significant amount of research has been concerned with the structure of training data, interpreting the mappings implemented by trained networks (leading further to rule-extraction), and how the network's resources are used by learning algorithms.

Elman [69, 70] applied PCA to the analysis of the internal representations of simple recurrent networks trained on grammatical tasks. The hidden unit responses to a set of data are used to construct the covariance matrix in this case. Dennis and Phillips [64] also discuss the use of PCA as well as Canonical Discriminant Analysis (CDA) to interpret the hidden unit activations of networks trained for a fixed number of epochs. CDA is another statistical dimensionality reduction technique which seeks an interesting projection of the data. However, in CDA each data point is assumed to carry a class label for one of a number of classes occurring in the data. The projection then aims to make the different classes as distinct as possible. When class data is unavailable, the authors suggest that PCA and CDA can be used together to provide a useful network interpretation tool.

Weigend and Rumelhart [245, 246] also perform PCA on the covariance matrix of the hidden unit activations. Unlike [64] however, the primary focus is on the effective number of parameters being used by the network to perform the task at hand. Consequently, they examine the evolution of the eigenvalue spectra as training proceeds. They also perform a SVD of the space spanned by the input-to-hidden layer weights. This information is compared with per-

formance on a validation set of data, and the point at which validation set performance is best is used to determine how many eigenvalues of the PCA/SVD are significant for learning. The results suggest that gradient descent extracts principal components successively, and that this can be used to explore over-fitting during training.

3.8.1 Training Algorithms and PCA

Some researchers have attempted to use PCA to address certain issues in the training of MLP's. Malki and Moghaddamjoo [148] have used PCA (KLT) to improve the training performance of backpropagation. Initially, PCA is performed as a pre-processing of the training data, and training is conducted using only the first few PC's as inputs to the network. Higher PC's are introduced as training progresses, the idea being to allow the network to learn the most important (in the PCA sense) features of the data first. Different learning rates are also used in the input layer weights, so that newly-added components are weighted higher than existing components. Results on a single training task suggest that this significantly decreases the computation time of backpropagation (in terms of the number of epochs needed to reach a certain given error value), and may improve the probability of reaching a lower error value. Note that batch-mode learning must be used, in order to perform the PCA transformation on the entire set of training data. No comparison is made in [148] with other techniques.

Taking a different approach, Levin et al. [140] propose a pruning algorithm using PCA. After a network is trained (e.g. using backpropagation), PCA is performed on the correlation matrix comprised of the input vectors to the first layer of weights. Principal components are ranked according to their influence on the linear outputs of that layer of nodes, and any components that do not increase the error on a separate validation set of training data are removed. Then, the weights in this layer are projected onto this reduced space, resulting in a reduced network. This process is repeated for each layer in the network. Results show improved accuracy on function approximation and time series prediction tasks. Other techniques, such as weight decay and early stopping, are also used in conjunction with the PCA pruning, to further improve results.

Finally, a similar pruning technique is developed in [122]. The authors here highlight the computational efficiency of using the SVD rather than solving the eigensystem of the correlation/covariance matrix. The relevance of PC's/singular values to layer output values is de-

termined using the related QR column pivoting factorization method (see also [112, 117] for discussion of computation issues and SVD).

Note that each of these PCA techniques is independent of the learning algorithm, in the sense that virtually any algorithm could be used in place of backpropagation. It is also possible to use other techniques in conjunction with PCA (Levin et al. have done this with weight decay and early stopping [140]). Unfortunately the interaction between these different issues is far from clear. A large experimental study would provide some information about significant practical issues in this regard. This issue and others related to the comparison and testing of MLP training algorithms are discussed in Section 2.9.

3.9 Summary

This chapter has examined the area of scientific visualization methods for the analysis of neural networks - specifically the MLP training methods and the error surface. In particular, the technique of PCA has been used for visualization of the learning process in MLP's. This technique allows visualization of the trajectory followed during learning in networks of practical size, where the dimensionality of the search space is often very high. These results suggest that learning trajectories such as those produced by backpropagation can be represented in a low dimensional subspace to a high degree of accuracy.

The temporal evolution of the PC's and PV's has also been explored. This investigation reveals information about the dynamics of the path taken through weight space by the given training algorithm. It is also possible to observe the evolution of the mapping produced by the network, and to interpret different aspects of this process (e.g. the role which is played by a particular hidden unit).

These PCA visualization techniques are independent of the learning algorithm used, the training set, and the various adjustable parameters of the learning process. PCA can be performed in a computationally efficient manner using SVD, hence it scales up with the size of the problem under consideration. Data is also inexpensive to collect, being generated automatically as part of training, without requiring additional information (e.g. class/output values).

Chapter 4

The Nature of the Error Surface

This chapter explores in detail the nature of the error surface in Multi-Layer Perceptrons (MLP's). Previous research providing numerous results, ideas, theory and practical experience with MLP error surfaces is summarized and discussed.

There has been some interest in the structure of the "configuration spaces" of other complex systems, mainly in the fields of combinatorial optimization problems, statistical physics and evolutionary algorithms. Work in these areas is described, and some of the techniques developed in these other areas are adapted to the analysis of MLP error surfaces. These techniques favour a methodology for examining the properties of MLP error surfaces via statistical sampling methods.

Ultrametric structure is a third-order statistical property (i.e. computable on triples of data points) which has been detected in statistical physics systems and combinatorial optimization problems. In this chapter, ultrametric structure is also detected in the low-lying regions and apparent minima of the error surfaces of MLP's in a number of experimental situations.

4.1 Analytical Properties

Since the initial development of the backpropagation algorithm, many researchers have studied the error surface in an attempt to understand its properties and to relate this to the performance of different learning algorithms and to the complexity of MLP mappings. This research has resulted in a large amount of diverse results that have not been reconciled and which often apply to specific constrained learning scenarios.

It is possible to summarize the main motivations for investigating the properties of the error surface:

- It assists in the adaptation of traditional optimization algorithms to the application of training MLP's.
- It aids in designing new algorithms and heuristics (e.g. according to natural metaphors).
- It aids in the explanation of how algorithms/heuristics operate.
- The surface metaphor, despite also sometimes being a source of confusion, provides intuition through analogy with the features of a 3-D surface.
- It leads to numerically testable definitions for encountered training conditions.

As discussed in Chapter 2, it is clear that the error surface is not completely arbitrary, in the sense of the No Free Lunch (NFL) theorems and a black-box optimization model. Instead, the error surface is a representation of the combination of a particular supervised learning problem, error function and an MLP network topology (Figure 2.4).

In this section a number of results concerning MLP error surfaces are discussed. This includes results on restricted networks and training problems, analytical results, heuristics and practical experience.

4.1.1 Linear networks

An MLP network can be simplified by using linear activation functions on all hidden and output nodes. Such a network implements a linear mapping from its input to its output space. A linear MLP with an arbitrary number of hidden layers can be reduced to a single layer of weights (i.e. no hidden layers), through a linear transformation. Although the approximation capabilities of such networks are restricted, they remain of interest to Artificial Neural Network (ANN) research and application for a number of reasons (see [15] for a review).

Baldi and Hornik [14, 15] have proven that for a linear MLP network, the error surface has a unique global minimum, and that all other critical points are saddle points. This result holds for any number of hidden layers. The error surface is not convex in this general case, but for the simpler case with no hidden layers (i.e. a single layer of weights), Baldi and Hornik show

that the error surface is strictly convex [15]. The authors also obtain results for linear autoassociative, or *autoencoder* networks, which are related to the calculation of principal components. These results are discussed in Section 4.1.3.

4.1.2 Exclusive OR (XOR)

The XOR problem is one of the most commonly used training tasks for MLP's in the literature. XOR is a very simple problem to state, yet it is an example of a linearly inseparable classification task. Minsky and Papert used XOR to highlight the limitations of single-layer perceptrons [152]. In response, Rumelhart et al. demonstrated the effectiveness of using backpropagation to train an MLP to solve XOR [203].

Two "minimal" MLP architectures have sufficient capabilities to learn the XOR problem. A standard MLP requires two hidden units (2-2-1 XOR). However if shortcut connections are allowed directly between the input and output layer, one hidden unit is sufficient (2-1-1 XOR). Although XOR was the exemplar non-linear task used to demonstrate the ability of MLP's to learn non-linearly separable patterns, it is not completely trivial to train an MLP to learn XOR. For backpropagation, successful convergence depends significantly on the values chosen for the parameters, such as the learning rate, momentum, initialization range for weights and update mode (batch or on-line).

It has sometimes been claimed that the error surface(s) of the 2-2-1-MLP and 2-1-1-MLP XOR problem contain local minima [35, 106, 142]. This is reflected in the common perception in the literature that backpropagation can become trapped by local minima when trying to solve the XOR problem [82, 203]. Rumelhart et al.'s original interpretation of the occasional failure of MLP's to learn simple problems including XOR was that this was caused by local minima. This led to a widespread perception that local minima were the key issue as far as MLP error surfaces were concerned [203]. The issue of local minima and the causes of bad training performance are discussed further in following sections.

Despite these results, careful analysis of the problem has eventually revealed that the error surface of both the 2-1-1 [224] and the 2-2-1 [87, 88, 89] networks have *no* local minima. All stationary points in the 2-1-1-XOR problem are saddle points. Note that while the usual definition of local minima is generally accepted (Section 2.6), it allows for the possibility of infinite weights, which has no clear meaning as far as the practical implementation of MLP's is

concerned. Hamey proposes alternative, intuitive definitions of different kinds of local minima which do not allow stationary points with weights at infinity [89]. He then shows that there exist finite trajectories which allow escape, without increase in error, for all finite stationary weight points, and hence no local minima. In addition, Sprinkhuizen-Kuyper and Boers have shown that all stationary points with finite weights are saddle points with positive error or absolute minima with error zero, hence no local minima occur for finite weight values [225].

Finally, methods of homotopy have been applied to the analysis of the 2-2-1-XOR error surface [49]. Homotopy is a continuous transformation from a known analytical function to another function (in this case the error function $E(\mathbf{w})$). A large number of local minima are found by this approach, through the addition of a regularization term to the error function. The practical implications of this result are not clear. Despite this, the work indicates that empirical MLP error surfaces have an extreme ratio of saddle points to local minima, which is agreement with much of the work discussed in this section [89, 225].

Overall, the analysis of the XOR error surface indicates that local minima are not the cause of poor training performance for algorithms such as backpropagation on this problem. Other features of the error surface such as saddle points and plateaus, seem more likely explanations of training difficulties.

4.1.3 Autoencoder Networks

Autoencoder networks (also called Autoassociative nets, or simply encoders) are another very popular type of problem in MLP research. An autoencoder network has an equal number of input and output nodes. Its task is to learn an identity mapping of a set of training patterns. The patterns are typically orthogonal, encoded in binary (or bipolar) values with one input value equal to 1 and all others equal to 0 (or -1). Autoencoders can have one or more hidden layers with the number of nodes typically smaller than the number of inputs/outputs. In this case, the network must develop some kind of compact internal representation in order to solve the problem to an acceptable degree of accuracy.

For an N_i-input/output encoder, it is clear that $\lceil log_2(N_i) \rceil$ hidden units are required to perform a binary (base-2) encoding of the data at the hidden layer (threshold functions could be used in this case). When the hidden units use intermediate activation levels, Kruglyak has shown that an $N-2-N$ encoder can be constructed that can solve the encoder problem for

arbitrarily large N_i [132]. The task of finding the appropriate weight values to obtain this mapping remains an intractable non-linear optimization problem. Backpropagation has been shown experimentally to be capable of finding a solution (in a reasonable amount of time) for $N_i < 8$ [143], and Quickprop [74] is feasible for $N_i < 13$ [144]. This work examines the position of hyperplanes in the 2-D hidden unit space during learning, and concludes that the error surface is "ill-conditioned" for gradient descent learning, due to large changes in the size of the local gradient. A solution to this problem is to change the encoding or input representation. Adopting this approach, Bakker et al. were able to solve much larger encoder problems using a block encoding[1] for output patterns [12, 13]. A $1000-2-1000$ encoder was thus trained using backpropagation, requiring $\approx 6.6 \times 10^9$ epochs. In any case, the encoder problem is convenient for experimental purposes because it can be scaled to any desired size, and the difficulty of the problem can be somewhat controlled.

A simple encoder is the linear encoder (which can be reduced to a single hidden layer, if a dimensionality reduction is to be performed), as mentioned in Section 3.5 above. For such a network, it can be shown that the error surface has a *unique* minimum [14, 15] corresponding to the projection onto the subspace generated by the first principal components (PC's) or "principal vectors" of a covariance matrix associated with the training patterns. A number of different techniques have been developed for the on-line computation of PC's via the training of linear encoder networks (see [65, 163] and the references therein). On-line computation is important when dealing with large, high-dimensional data sets, which have prohibitive memory requirements and computational difficulties. Bianchini et al. [31] have proven that local minima are possible in the more general case of a non-linear encoder.

4.1.4 Further Analytical Results

Despite the generality of the MLP training optimization problem, it is clear that an MLP is not necessarily to be considered as a "black-box" model. The weights in an MLP are related in certain definite ways, such as the fully-connected layered structure, and the way in which activation is propagated through the network as non-linear functions of weighted sums. A number of further general results concerning the structure of MLP error surfaces are summarized in this section.

[1] In the encoding used, equal (or near equal) quantities of 1's and 0's are present in the target vectors.

Weight-space Symmetries

The simplest analytical properties of MLP error surfaces are the so-called *permutation* and *sign-flip* symmetries of weight space [63, 92, 94, 93]. Considering the general MLP architecture shown in Figure 2.3, it can be seen that labeling of weights from the left-to-right and top-to-bottom in the network as $w_1, \ldots, w_n$, is completely arbitrary and simply for convenience. The geometrical idea of an N-dimensional weight space is independent of this labeling, thus these labels can be changed without changing the properties of the weight space (or the error surface) itself. This fact means that in a fully connected MLP, hidden units can be exchanged, including all ingoing and outgoing weighted connections (within the network), without changing the mapping implemented by the network (all that is required is a relabeling of the weights). This is referred to as the *permutation symmetry*. This symmetry means that any given weight vector is part of a set of N_h weight vectors that are functionally equivalent, in that the network mapping is identical for every weight vector in this set.

A second symmetry can be seen as a result of using an odd activation function

$$f(-x) = -f(x)$$

on hidden units, such as the $tanh$ sigmoidal function. This fact means that if the sign of every weight on the input and the output of a sigmoidal node is inverted, the network mapping will again be unchanged. This is referred to as the "sign-flip" symmetry. This property introduces a factor of 2^{N_h} symmetric weight vectors.

If the permutation and sign-flip symmetries are taken into account, then for almost any given point in weight space, the number of equivalent weight configurations (points in weight space) is $2^{N_h} N_h!$. This property was first investigated theoretically by Sussmann [229] and Chen et al. [41] (see also [95]). Sussmann shows that the mapping produced by a network is unique, apart from these symmetries, provided that the network is minimal (i.e. there are no redundant nodes in the network which make zero contribution to the mapping, such as a hidden node with a zero output weight). The symmetries of weight space allow the formulation of an open minimal sufficient search set as either a wedge or cone interior. In practice, points in weight space can be transformed into this "fundamental" wedge[2] by constraining the bias

[2]Including points on the boundaries of the wedge, and so this is not the true minimal sufficient search set.

weights of all hidden layer units to be positive, and the bias weights of each hidden layer unit to be ordered in magnitude[3] [41, 95]. This fundamental wedge of weight space occupies only a very small fraction of the total volume of weight space. Note also that weight vectors related by these symmetries have the same magnitude, hence the global minimum point gives rise to a "spherical throng of optima" in weight space [41]. Further work on weight space symmetries has extended to more general activation function conditions [134], and the formalism has been further refined [169, 202].

In practice, any implications of the symmetry results need to be considered. For the purposes of training an MLP using backpropagation or some similar kind of trajectory-following learning algorithm, Chen et al. [41] conclude that the symmetries of weight space have little consequence, because an algorithm cannot exploit this knowledge in any way; constraining the algorithm to lie within a unique area of weight space does not lead to better results than letting the algorithm wander out of such a region.

It is further conjectured [41] that a non-descent-type of algorithm, such as a randomized search procedure, will not benefit from weight space symmetry. This is because the constrained optimization of weights in a unique region of weight space has only one global minimum that it can possibly locate, whereas in considering the whole of weight space an unconstrained algorithm need only find one of $2^{N_h} N_h!$ equivalent global minima.

In fact it is possible to show that a trivial global optimization algorithm can benefit from knowledge of symmetries such as the sign-flip and permutation symmetries in the search space. Consider taking p samples uniformly at random over a given search space W, where $|W| \gg p$. On average these samples will be evenly distributed across the error surface. Because the algorithm does not use cost function or derivative information, its expected performance simply increases with p. If it is known that the search space is two copies of half itself, random search can cover the space twice as well by spending all of its p samples in one symmetrical half of the original search space. This is equivalent to allowing twice as many samples in the original search space. Thus, the expected performance of this algorithm must increase. This argument is analogous to that in discussing the improvement in performance for stochastic algorithms which do not revisit points [206].

Hence, it is not impossible that the symmetries of the MLP error surface can be use in a

[3]Given a sufficiently high precision for representing floating point numbers, the probability of having equal weights or a zero bias weight are negligible.

training algorithm to improve performance. It remains to be seen if this improvement can be demonstrated in practice, or more importantly if this benefit extends to allowing such a method to outperform other proposed training algorithms, which do not exploit this knowledge but which have shown very good training performance.

A further example of when the sign-flip and permutation symmetries can be taken into account is in the context of Bayesian learning methods. The symmetries are used in this case for comparing the performance of different supervised learning models (see [32] for details).

Local Minima

As stated above, the issue of local minima has often been the focus of MLP error surface research, and it has been shown formally that local minima can exist, even in very simple situations [222] (i.e. no hidden layer).

Auer et al. examine a single neuron network with N_i weights/inputs and k training examples [7]. They prove that for any composition of the error and activation functions which is continuous and has bounded range, the error surface can have up to $\lfloor k/N_i \rfloor^d$ local minima. A neuron using the mean-squared error (MSE) function and the logistic sigmoid is one example of this. In addition, any transfer function with bounded range can have exponentially many local minima with the MSE error function. This result assumes that the examples are non-realizable, that is, there is no weight vector for which the error value is zero. For the realizable case (with minimal assumptions on the error and activation) there can be no local minima, only the global minimum. They also show that for any bounded activation function and the squared-loss function, the error surface becomes flat as $||\mathbf{x}.\mathbf{w}||$ becomes large. This is not always true, for example in the case of a cross-entropy error function and the logistic sigmoid, the error surface turns flat only at the global minimum.

Further Results

A number of other results concerning the error surface have appeared in the literature. In considering the error function itself, Solla et al. show analytically that for a cross-entropy error measure, the error surface is steeper in the region of local minima than the usual quadratic error measure [220]. They further suggested that a useful empirical measure of the average steepness

(i.e. magnitude of the gradient) of the error surface might be

$$S \equiv \langle |\nabla E(\mathbf{w})| \rangle$$

where S is obtained by random sampling in weight space. This kind of technique is used in this chapter of this thesis to explore other statistical sampling-based properties of the error surface. Solla et al. use the above measure to compute a ratio of the gradient magnitudes S_L (logarithmic/entropy error) to S_Q (quadratic error) of $\simeq 3/2$, but the experimental details for this ratio are not given.

Poston et al. [180] offer a proof that if there are as many hidden nodes as patterns then almost certainly[4] a solution exists, and the error function has no local minima (using a sum-of-squares error (SSE) function). In practice however, an algorithm such as backpropagation even in this situation is still susceptible to saddle points, plateaus in the error surface, and associated finite learning rate and numerical precision problems (see next Section).

For an MLP trained with backpropagation, results are also known concerning the convergence of backpropagation to the optimal solution. For batch backpropagation, this convergence was shown by Gori and Tesi to be guaranteed for linearly separable training patterns [82]. This is dependent on the learning rate value and assumes a pyramidal network structure (i.e. from the hidden layer through to the output layer, the number of neurons cannot increase). This assumption has been relaxed, as well as allowing multiple hidden layers in the network [83]. For on-line backpropagation, a similar result can be shown, independent of the learning rate [81].

4.2 Empirical Results, Experience and Heuristics

The analytical studies concerning the properties of MLP error surfaces discussed above provide some important insights. It is also apparent that these insights are limited, and sometimes apply only in restricted learning scenarios that are not realistic. The large amount of practical work done in the field of learning in MLP's has led to a number of empirical insights which may not hold in any rigorous sense, but were obtained in real training situations.

[4] Excluding only some contrived situations.

4.2.1 The Importance of Local Minima

The most well-known difficulty that arises in general optimization problems is the issue of local minima. Mathematical programming and optimization research was originally concerned with univariate problems (finding the minimum of a function of a single variable), or with solving systems of equations or inequalities involving only a few variables. In the one-dimensional case, the concept of a local minima follows closely from the issue of convexity. The conceptual picture is that if there are no local minima, then the optimization problem is trivial, and the cost function resembles a parabolic bowl or single valley.

This picture has persisted in MLP research, perhaps mainly because it was used to explain the failure of backpropagation to learn, and because of the large amount of techniques from optimization being applied to the development of better training algorithms. A common line of thought was that if training was successful (by some criterion), then the algorithm was finding the global minimum, whereas if training did not progress satisfactorily (within the constraints of the experiment) then the algorithm was stuck in a local minimum. As a result, it was considered that local minima were the main obstacle preventing more successful applications of MLP's.

The results of the previous sections indicate that local minima certainly exist in some MLP error surfaces, including some very simple cases. Note however that some of these examples have been specially constructed to show this existence. Others include local minima at points involving infinite weights, due to the asymptotics of sigmoidal functions, which are less interesting from a practical viewpoint.

Crane et al. [55] examine a number of MLP's using a student-teacher learning model (Section 3.6.1), and use a quadratic programming method to approximate the number of local minima on the error surface. For a fixed network topology, the number of local minima is found to be a decreasing function of the ratio of training set size to the number of weights in the network. When the training set size is less than the number of weights, there is a low probability that training from a random starting position followed by refinement leads to a local minimum. For a fixed training set size, the number of local minima increases with the network size. These results are interesting, however the particular local minima obtained are directly related to the algorithm used (Sequential Quadratic Programming with local refinement). The networks used were also quite restricted (3-1-1 up to 3-5-1 configurations).

There is however evidence to suggest that local minima are not as big a problem for learning

as first perceived. In fact they may be relatively rare in practical MLP error surfaces [94]. It is intractable to verify this notion empirically, and no theoretical results are available as local minima are dependent on the training data. Approximate empirical studies may be able to provide more insight into quantifying the occurrence of local minima. Some results concerning high-dimensional spaces (see Section 4.4.1), may eventually lead to theoretical insights into this issue.

It seems that while local minima are often discussed in describing the actions of training algorithms and other observations in practical scenarios (particularly in the early literature), often the observations in question are due to factors other than local minima. These factors are discussed in the following section.

4.2.2 Ill-Conditioning and Properties of the Gradient

The results and experience of research into the properties of the error surface have identified an important feature of MLP error surfaces which has implications for successful training. Broadly speaking, the presence of relatively steep and flat regions is a fundamental feature of the error surface.

The rate of change of the gradient, that is properties of the second-derivatives (Hessian matrix) of the error surface determines its relative steepness or flatness. If all the eigenvalues λ_j of the Hessian are equal, the Hessian is a diagonal matrix, corresponding to an error surface which is (locally) a perfectly circular bowl, with cross-sections of the error surface being hyperspheres in N-dimensional space. A gradient descent algorithm with step size $\eta = 1/\lambda_j$ will reach the minimum in a single step. When the eigenvalues have different values, the error surface assumes a locally elliptical shape, with the slope in some directions being greater than in others. The behaviour of a gradient descent algorithm depends on how well-matched η is to the λ_j. Too large a step size will result in oscillations across the contours of the surface and may cause divergence. On the other hand, too small a step size will make convergence to the minimum very slow [57, 138].

Clearly, the *condition number* of the Hessian (i.e. ratio of eigenvalues $\lambda_{max}/\lambda_{min}$) is a significant factor in how different training algorithms negotiate the error surface. Algorithms that do not use gradient information directly, will be affected implicitly through their reliance on the values of the error function, which will vary according to this ratio. Algorithms such as

Quasi-Newton (QN) and Levenberg-Marquardt, which use second-order information, may not converge much faster than gradient methods in such a situation, and due to their increased computation effort may actually result in slower execution times [139]. When this ratio is "large", the Hessian/error surface is said to be ill-conditioned.

It is known that MLP error surfaces are often ill-conditioned [57, 204], with eigenvalues differing by orders of magnitude. Relative to each other, this fact means that there are often directions on the error surface in which the gradient varies quickly (cliffs or steep ravines) and others where the gradient variation is quite slow (plateaus or flat regions). This principle is certainly in agreement with empirical evidence and practical experience with MLP training [93, 139]. For an algorithm such as backpropagation with a single fixed step size η, this feature leads to periods of very slow progress, sudden drops and oscillations in the error value attained as a function of training iterations. Effects such as premature saturation of output units can sometimes be thought of as the result of the training process moving to a very flat region of the error surface [241].

There are several factors that contribute to the ill-conditioning in MLP error surfaces. The properties of sigmoidal activation functions are undoubtedly reflected in the properties of the error surface [23], with superpositions of sigmoids leading to further ill-conditioning [204]. Statistical features of training data, such as biased mean values, unequal variances of input or desired output variables and linear correlations can also have ill-conditioning effects. Attempting to make sure the sigmoids in the network operate effectively over their useful region is one way to reduce the effects of ill-conditioning [138]. Very small training sets may also contribute to ill-conditioning [150].

The results on ill-conditioning in MLP error surfaces highlight the importance of this kind of research in the pursuit of more effective training strategies. Many of the problems encountered in training can be attributed to ill-conditioning. While local minima are important from a theoretical view, and may eventually have implications for understanding the fundamental properties of the error surface, ill-conditioning seems a more much direct and practically important effect for the study of training algorithm performance. The presence of local minima on an error surface may be quite insignificant in terms of its effects on training, compared to the degree of ill-conditioning on the error surface.

4.2.3 What does the Error Surface "look like"?

The structure of the error surface has been a subject which has concerned researchers using MLP's since they became popular following the backpropagation work of Rumelhart et al. in the mid-1980's [203].

Denker et al. present the following description of the error surface:

> *These symmetries imply a certain periodicity in* $\mathbf{w}$ *space. This leads us to visualize the* $E(\mathbf{w})$ *surface as resembling a sombrero, or as a phono record that has been warped in certain symmetric ways: near the middle* $(\mathbf{w} = 0)$ *all configurations have moderately bad E values. Radiating out from the centre are a great number of ridges and valleys. The valleys get deeper as they go out, but asymptotically level out. In the best valleys, E is exactly or asymptotically zero; other valleys have higher floors. This picture is, of course, an oversimplification.* ([63], p. 887)

This description is based on consideration of the permutation and sign-flip symmetries of weight space, as well as the effect of the sigmoidal functions on the surface. A symmetry about the origin is implied, with a surface as $||\mathbf{w}|| \rightarrow \infty$ comprised of flat regions at different values of E, connected by step-like transitions (as the sigmoids operate at their saturation points for most sums of very large weights).

Hush et al. have studied MLP error surfaces for two different networks [110, 111]. In the first case, a 2-weight network is used, consisting of a single unit with input weight and one bias weight (see Section 3.4.2). This simple net allows the complete error surface to be viewed. The training task is to distinguish between two different classes (outputs) for an integer-valued input. These error surfaces have a stair-like appearance, with a number of different levels or plateaus extending out from the origin. For a linearly separable dataset, the stair-structure is symmetric, whereas for a more complex dataset, some levels begin to dominate the surface. Others seem to collapse, with ridges and narrow valleys forming, and extending from the origin outwards. These observations are in agreement with the examples presented in Section 3.4.2 above. Phillips has also shown that equal classes of examples can lead to a more symmetric error surface [175]. The same two-weight network is again used to demonstrate this, and the hypothesis is verified on another larger problem.

The second problem considered by Hush et al. is an artificial two-class classification problem, where each data point from each class is drawn from a separate 4-dimensional Normal

distribution. A 4-2-1 MLP was used for this task, having 13 weights including biases. Backpropagation was used to find an apparent minimum on the surface, and 2-D slices of the error surface were visualized by holding 11 of the weights at this apparent minimum value. The slices reveal similar surfaces with the characteristics of a large number of flat regions populated by a number of narrow valleys and sharp transitions. The error surface for a small training set (10 points) displays these characteristics most predominantly. Hush et al. increase the amount of data, and observe that while these features persist, the error surface becomes generally smoother and curved in many places.

To examine the variation of the gradient, histograms of the magnitude of the gradient are presented [110], collected from samples in a quadrant of the error surface not containing the minimum[5] for the 2-weight error surface. The gradient varies over approximately 7 orders of magnitude, with most of the distribution concentrated in about 3 orders of magnitude. It is also shown analytically that as the weight values approach infinity, the magnitude of the gradient approaches zero [110].

Little research has attempted to examine properties of the error surface as a function of the topology of the network. Practical experience has shown that algorithms such as backpropagation often experience more difficulty in training MLP's with multiple hidden layers, due to the tendency for values of the Hessian to be smaller for layers of weights near the input and larger for layers near the output [138]. Networks with multiple hidden layers may also be more sensitive to weight initialization values [62].

4.2.4 Initialization of Weight Values and the Error Surface

Rumelhart et al. [203] observe that if all weights in an MLP start with equal initial values prior to training, backpropagation will fail. The reason is because the error signals that are back propagated through the weights are in proportion to the weight values themselves. Points where all weights are equal therefore represent a line of stationary points on the error surface [203]. Rumelhart et al. avoid this problem by initializing the weights to small random values prior to training. This is referred to as *symmetry breaking*. These early experiments with backpropagation revealed that random initial weights led to learning, though repeating the training process several times often led to different final results.

[5] In an attempt to avoid "characteristics of the surface that are only encountered near the minimum" [110].

From an optimization viewpoint, this process can be thought of as one of the simplest "global" optimization schemes - repeated application of a local search method from different starting positions on the error surface [237]. However, in the case of MLP training, this issue has attracted a large amount of attention in the literature (see [233, 234] for surveys). It is beneficial to consider the reasons for this interest.

Initialization of the weights to small random values was found to be reasonably effective in practice. In the absence of other prior knowledge, there seems to be no reason to prefer any particular point in weight space as a starting point for training than the origin. This choice also avoids having to define a feasible (bounded) region of values for weights, whereas some other optimization problem might proceed by initializing uniformly over the entire search space. It has also been observed that initializing with large weight values frequently leads to rapid saturation of some of the sigmoidal activation functions (see e.g. [131]), as discussed above.

Studies have shown that the success of training algorithms such as backpropagation can be highly sensitive to the values of the initial weights [130, 207]. By systematic variation of the initialization points, Kolen and Pollack show very complex, fractal-like patterns of areas of convergence and non-convergence for the simple XOR problem [130].

The initialization of MLP weight values is one example of how heuristic and prior knowledge can be incorporated into the MLP training problem (Section 2.7.4). Knowledge of the sigmoidal function, and the manner in which derivative information is calculated, have led to heuristics which are widely used and often work [138].

Finally, it is worth remembering that while initialization close to the origin is by far the most widely used approach, this choice is somewhat tied to the assumption that training is based on derivative-based methods (see, e.g. [138]). Alternative heuristics and methods are certainly possible (e.g. [146]).

4.2.5 High-dimensional Spaces

MLP networks typically contain hundreds of weights. This fact means that the error surface is a surface of very high dimensionality. Although the surface metaphor is used to help visualize the search space and the workings of optimization algorithms, mathematical studies have shown that human intuition, based on the geometry of 2-D and 3-D space, is often not applicable to high-dimensional spaces.

Firstly, consider a the volume of an N-dimensional hypercube and a corresponding hypersphere inscribed inside the hypercube. As N increases, the ratio of these volumes approaches zero [93, 95, 211]. This result means that for high-dimensional hypercubes most of their volume is contained in the corners. The Euclidean distance of these corners from the center of the hypercube grows without bound as $N \rightarrow \infty$. This fact creates a view of a hypercube as a rather unusual shape with a large number 2^N of very long spines [93]. Next, consider the sphere of radius $r - \varepsilon$ inscribed inside a larger sphere of radius r. As N increases, the volume of the outer sphere becomes increasingly concentrated in the shell between the two spheres [116]. A consequence of this result is that for uniformly or normally distributed data centered on the origin, the mean and variance of the *distance* of the data from the origin increases as a function of N.

Another result concerns the angle between a diagonal and the coordinate axes. As N is increased, this angle asymptotically approaches 90°, meaning that the diagonals become almost orthogonal to the Euclidean coordinates [95, 116]. Other non-intuitive examples have been described; for example the tendency of low-dimensional projections of high-dimensional data to be normally distributed [116] and the fact that two 2-D planes in 4-D space may have only one common point [93].

It is important to remember these results when thinking about MLP error surfaces. Everyday geometrical concepts may not scale up to high-dimensions, and this idea should be borne in mind when employing visualization techniques and designing training algorithms. In particular, the effects of ill-conditioning can be much more severe in higher dimensions [154]. Consider a highly ill-conditioned 2-D elliptical bowl, which produces a long, narrow ravine-like shape. In the direction of the smaller eigenvalue, the surface slopes gradually toward the minimum, whereas in the direction of the larger eigenvalue, the descent is very rapid. In high-dimensional space, there may be many orthogonal directions with relatively small slope, and only one or two directions with relatively large slopes. The direction for most efficient descent is therefore much more difficult to determine, as a consequence of the dimensionality of the surface.

4.2.6 Summary

The results discussed in this section provide a description of the structural features or properties of the error surface in MLP training situations. The error surface is a high-dimensional search

space which is everywhere differentiable and non-negative. Symmetries exist and produce a high degree of redundancy on the error surface, such as spherical throngs of stationary points. Local minima are not a major feature of MLP error surfaces in the sense that is often perceived from optimization problems in other areas. The main geometrical features are large, flat regions, some asymptotically approaching infinity, as well as step-like transitions, narrow valleys and ridges. The Hessian matrix is often ill-conditioned, meaning that the presence of flat and steep regions of the error surface is very prominent. These changes of gradient may vary over several orders of magnitude. The effects of ill-conditioning provide a satisfactory explanation for many of the difficulties associated with backpropagation and other training algorithms. More complex structure in the error surface can be thought of partly as the superposition of a number of these features.

4.3 Sampling Statistical Properties of the Error Surface

In this section, a number of experimental techniques which involve the exploration of the structure of the error surface via random sampling are discussed and demonstrated. Similar techniques have been used occasionally in the literature for the analysis of other search spaces or optimization problems, but rarely in the examination of MLP error surfaces. However the methods have a number of advantages which are discussed below. Most importantly, they are useful in exploring the error surface structure for practical sized networks.

4.3.1 Sampling Distributions of Error

To provide a description of an error surface, perhaps the most obvious question to ask is "what kind of error values exist on the surface?". One approach to answering this question is to examine the distribution of error values on the surface, using random sampling in some region of interest (e.g. [124, 221]). Since this only requires repeated evaluation of the error function, it can be done quite easily and efficiently. Figure 4.1 shows error histograms for a number of the training problems considered in this thesis - the 2-2-1 XOR, 4-2-4 and 8-2-8 encoder, 51-15-2 credit card, 9-5-2 cancer, 8-5-2 diabetes and 9-9-6 glass problems. Each histogram was generated from a sample of 10^4 points on the error surface, with each weight value drawn independently from the $\mathcal{N}(0, 1)$ distribution.

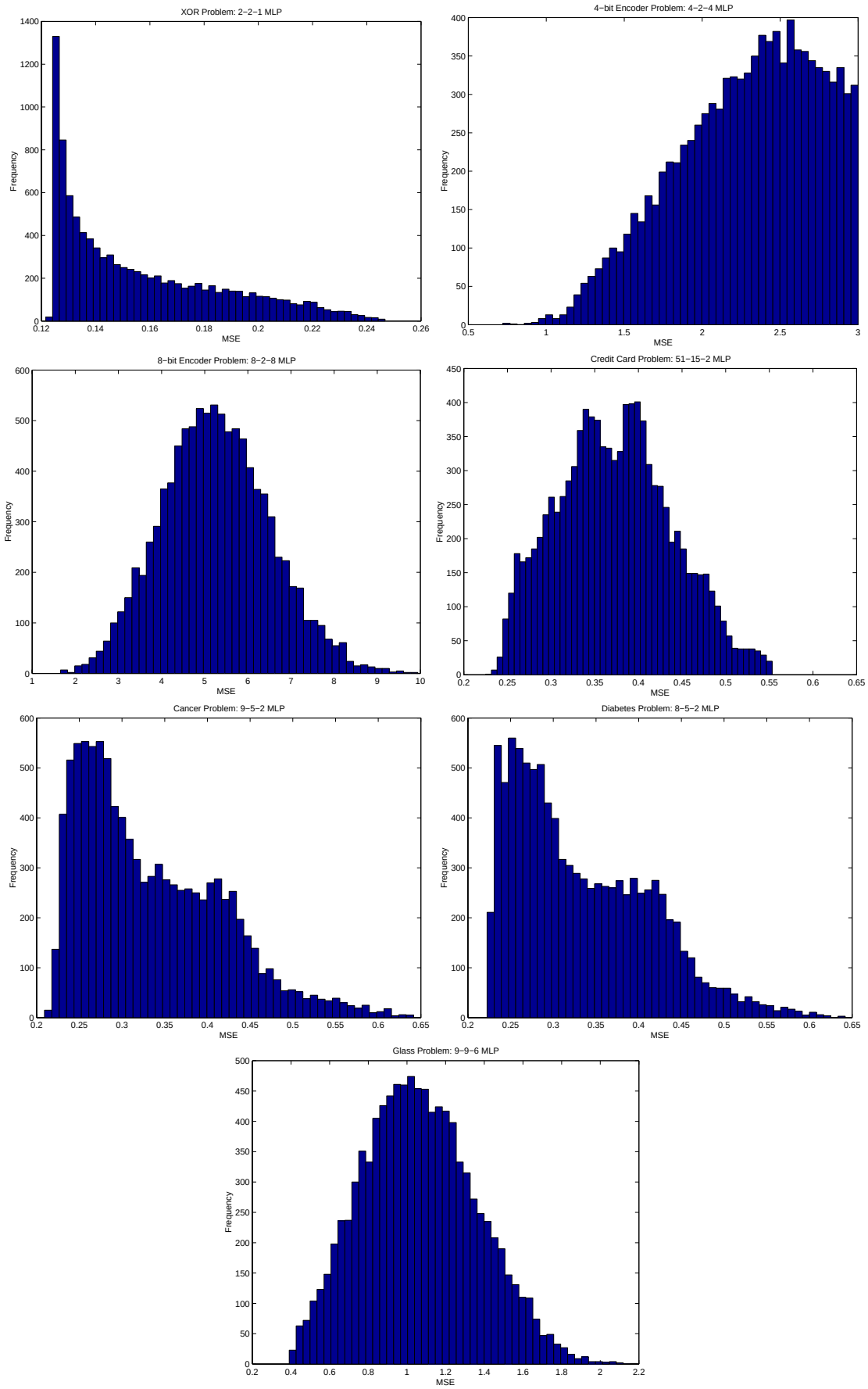

Figure 4.1: Error Histograms for several MLP Error Surfaces.

A number of observations can be made from these error histograms. Firstly, the distributions are quite different in shape from problem to problem, reflecting the dependency of the error surface on the training set and on the corresponding network architecture used. Secondly, many of the distributions are skewed left, rising quite rapidly and then falling of more slowly moving from left to right. This result gives an indication of the difficulty of finding a very low error value on the surfaces. The 4-bit encoder network is an exception, with a left tail extending toward more favourable error values. The 8-bit encoder is the only distribution which is very symmetric, with the glass problem also normal in shape (but noticeably missing the left tail). The XOR, cancer and diabetes error distributions resemble a log-normal distribution. These histograms give an indication of the amount of effort required by random search to locate a point with some given error value. For example, if the left tail decays roughly linearly, the time required to find a lower error value also grows linearly (assuming successively lower error values actually exist). By this rough measure, the XOR, cancer and diabetes surfaces can be expected to be more difficult for random search.

4.3.2 Distributions of Apparent Minima and Low-lying Regions

One can modify the above error distribution procedure by processing the samples in some way, so as to concentrate on a particular set of points on the error surface [36, 155, 221]. The author is aware of only one study by Schmidt et al. that displays results of this kind [207]. In this paper, the MSE histogram obtained with 1000 repeated runs of backpropagation from random initializations is shown after 1000, 4000, 20000 and 200000 epochs. A single artificial training problem is considered. Two other possibilities are considered in this section for selecting samples of interest from the error surface - these samples are referred to as *apparent minima* and *low-lying regions*.

Experimental Details

One set of points which is of interest are the local minima of the error surface. In combinatorial optimization problems, local minima can be located precisely given sufficient computation time. In the continuous regime, this task is infeasible. A compromise is to run backpropagation repeatedly with a small learning rate for a significant number of iterations, to obtain points which are “close” to minima or stationary points of the error surface. This method makes no

attempt to distinguish between regions such as flat plateaus and true critical points. The points collected at the end of training runs are referred to here as *Apparent Minima* (AM).

The AM samples are obtained from 4-bit and 8-bit encoder MLP's. Standard batch back-propagation was used with $\eta = 0.1$. At the end of each training run (30000 epochs), all weight vectors were transformed to lie within a unique wedge of weight space (Section 4.1.4), to remove the permutation and sign-flip symmetries of the error surface. Data samples consisted of 1000 points.

An alternative approach is to gather samples of *low-lying* areas of the error surface. Points are chosen by selecting each weight independently from a uniform distribution in the range $[-10, +10]$, restricting attention to the corresponding n-dimensional hypercube. The cost function is evaluated at each sampled point and points are rejected for which the cost function is greater than some fixed value. Denoting the ratio of accepted points to total number of points trialed by γ, the error threshold can be varied from $\gamma = 1$ (meaning all points accepted) through to a diminishing fraction of points accepted as γ decreases. There is no requirement for these points to be minima - the interest here is in low-lying areas of the error surface (where γ determines the level of "lowness"). For each value of γ, 100 sample points were collected.

Error Distributions of Apparent Minima

Given a sample of AM and their corresponding error values, the cumulative frequency distribution of the different error values in the sample can be examined (results for several encoder networks are shown in Figure 4.2). The first prominent feature for many of the networks was the step-like nature of the curves, indicating that a small number of error values often dominate a sample. Secondly, the curves shift upwards and to the left as the number of hidden units increases, indicating an increasing chance of an AM being an (increasingly) good solution ($E \approx 0$).

Distribution of Distances between Weight Vector Samples

Consider the probability distribution $P(q)$ of selecting any two points at random on the error surface, where

$$q = d(\mathbf{w_a}, \mathbf{w_b})$$

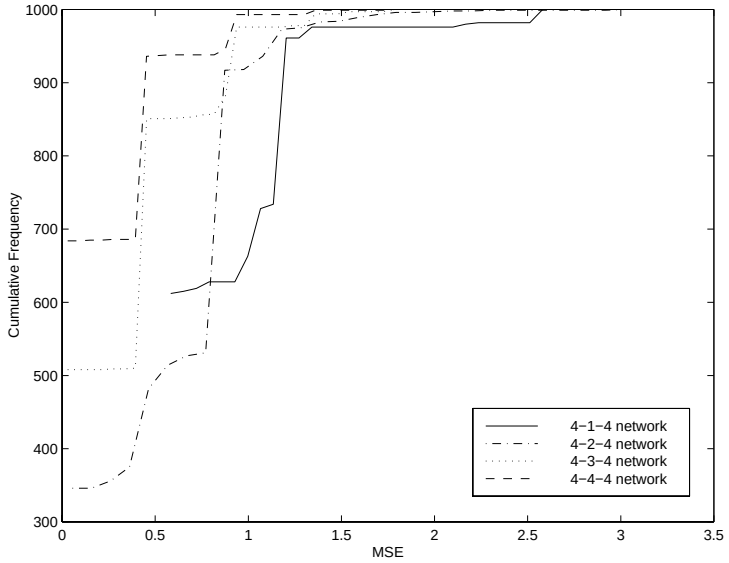

Figure 4.2: Cumulative error distributions for AM samples for the 4 bit encoder networks.

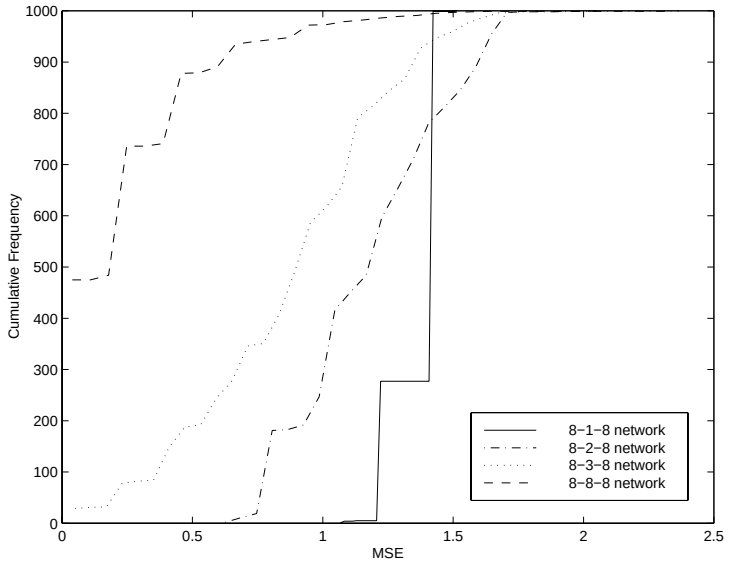

Figure 4.3: Cumulative error distributions for AM samples for the 8 bit encoder networks.

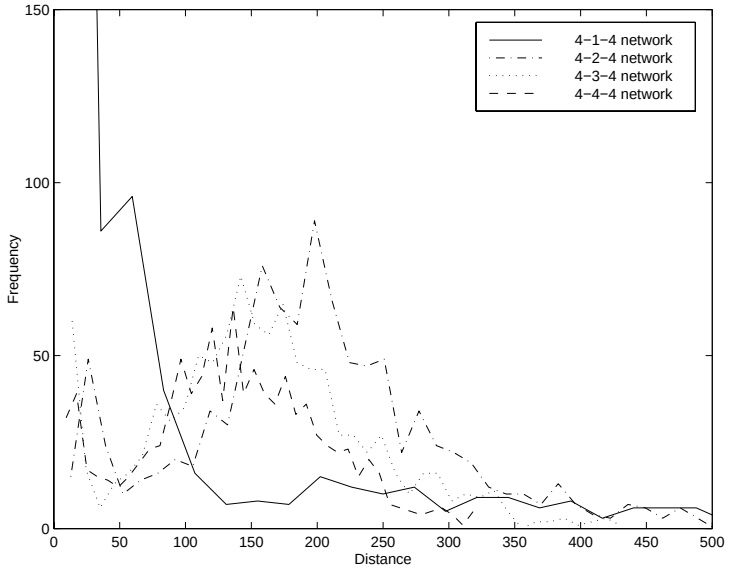

Figure 4.4: Sample distribution of distances between AM for the 4 bit encoder network.

is the Euclidean distance between points $\mathbf{w}_a$ and $\mathbf{w}_b$ [221]. This distribution can be used to obtain information concerning how a sample of weight vectors are arranged on the error surface, with respect to the distances between them. Note that as a consequence of the Central Limit Theorem this distribution will be approximately Normal for a sample of weight vectors sampled uniformly or from a normal distribution over some region of the weight space. This fact is independent of the structure of the error surface.

A sample of apparent minima, or a sample of weight vectors from low-lying regions of weight space can be used to generate a sampling distribution of $P(q)$, by choosing two points at random from the sample, and calculating the distance between them. Experiments were conducted on AM samples for the 4-bit and 8-bit encoder problems, and on low-lying samples for the cancer, glass and diabetes datasets, with varying network configurations.

For the encoder problems, the $P(q)$ distribution of AM is shown in Figures 4.4 and 4.5. Many of these distributions were skewed to the left (especially for the 4-1-4, 8-1-8, 8-2-8 and 8-3-8 networks). This observation suggests that a degree of clustering is present in the AM. Further examination is required to determine the nature of the clustering (e.g. the number of clusters). These histograms (Figures 4.4 and 4.5) are not good approximations to Normal distributions, indicating the samples are distributed somewhat differently to points selected at random from these error surfaces.

The results for the low-lying sample experiments are shown in Table 4.1. The first four columns show the breakdown of experiments in terms of the dataset used, the number of patterns

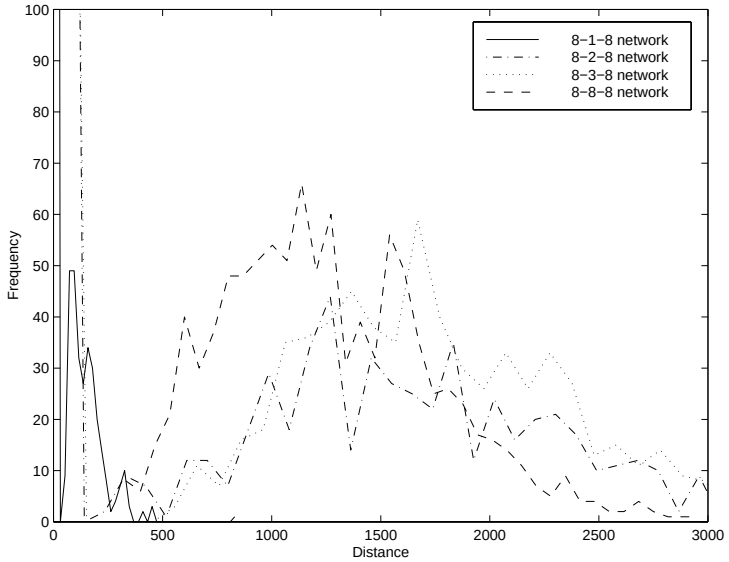

Figure 4.5: Distribution of distances between AM for the 8 bit encoder network.

in the dataset, network topology and sampling criteria (the mean squared error below which points were accepted, with the sampling ratio γ shown in brackets) respectively.

The mean and standard deviation of $P(q)$ for the low-lying regions of the error surfaces of the real-world problems are shown in Column 5 of Table 4.1[6]. Many of these experiments showed no observable trends in the histograms for the values of γ tested. This observation means that for those error surfaces, low-lying areas are distributed (for the statistics chosen) in roughly the same way as randomly chosen points. These histograms are approximately unimodal, symmetric about their mean values and are well-described by mean and standard deviation statistics.

One exception was the experiment on the cancer dataset using a 9-1-2 network (see Figure 4.6) In Figure 4.6 the distribution changes from unimodal to bimodal as the data moves from random to low-lying points, and this trend becomes more pronounced as γ is decreased. This observation indicates a clustering of low-lying areas into two regions, corresponding to the two different peaks of $P(q)$.

A different trend is observed in the experiment on the glass dataset using a 9-1-6 network (Figure 4.7). The distribution of $P(q)$ remains unimodal but moves to the left for $\gamma < 1.0$. This result suggests that distances between low-lying areas are distributed closer to each other than distances between random samples. This shift was observed to a small extent in several other

[6]The meaning of the last column of Table 4.1 is discussed in Section 4.5 below.

Dataset	**No. Patterns**	**Network(N)**	**Error Level(γ)**	$\mathbf{P(q)}$ $\mu(\sigma)$	**Correlation** s
Cancer	699	9-1-2 (14)	(1.0)	30 (4.8)	0.396
			0.25 (75)	29 (5.4)	**0.477**
			0.2 (1498)	30 (7.9)	**0.739**
			0.15 (11098)	30 (9.9)	**0.877**
		9-5-2 (62)	(1.0)	64 (4.7)	0.331
			0.23 (120)	65 (4.8)	0.344
			0.17 (1585)	65 (5.2)	0.358
			0.13 (8189)	65 (5.3)	0.367
		9-9-2 (110)	(1.0)	86 (5.1)	0.343
			0.23 (115)	85 (4.9)	0.336
			0.17 (1457)	86 (5.1)	0.353
			0.12 (8713)	86 (5.2)	0.353
Glass	214	9-1-6 (22)	(1.0)	39 (4.8)	0.377
			0.48 (131)	34 (4.8)	**0.433**
			0.42 (1467)	34 (5.1)	**0.436**
			0.387 (12817)	35 (4.9)	0.389
		9-7 6 (118)	(1.0)	89 (4.8)	0.338
			0.49 (132)	88 (4.8)	0.314
			0.44 (916)	87 (4.8)	0.336
			0.4 (12428)	87 (4.9)	0.344
		9-9-6 (150)	(1.0)	86 (5.1)	0.337
			0.49 (154)	85 (4.9)	0.364
			0.44 (1238)	86 (5.1)	0.325
			0.4 (16578)	86 (5.2)	0.354
Diabetes	768	8-1-2 (13)	(1.0)	29 (4.9)	0.410
			0.032 (131)	28 (5.2)	**0.449**
			0.022 (1061)	27 (5.3)	**0.483**
			0.0187 (11170)	27 (5.5)	**0.498**
		8-5-2 (57)	(1.0)	61 (4.6)	0.342
			0.045 (98)	61 (4.9)	0.390
			0.03 (1254)	60 (5.2)	0.390
			0.023 (14437)	60 (4.8)	0.368
		8-8-2 (90)	(1.0)	100 (4.8)	0.351
			0.046 (125)	99 (4.9)	0.369
			0.035 (1019)	99 (4.9)	0.345
			0.027 (11842)	98 (4.8)	0.377

Table 4.1: Summary of experimental configurations and results for real-world datasets.

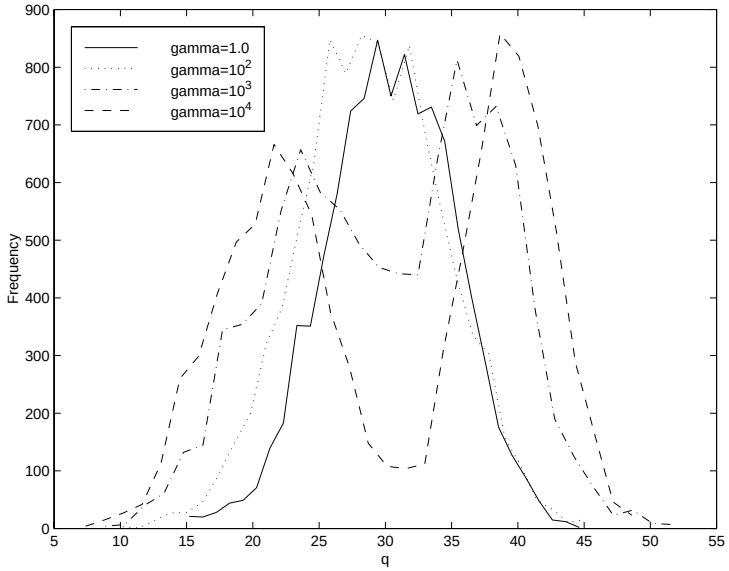

Figure 4.6: Probability distributions for the distance between two points in a sample for the cancer (9-1-2 network) experiment.

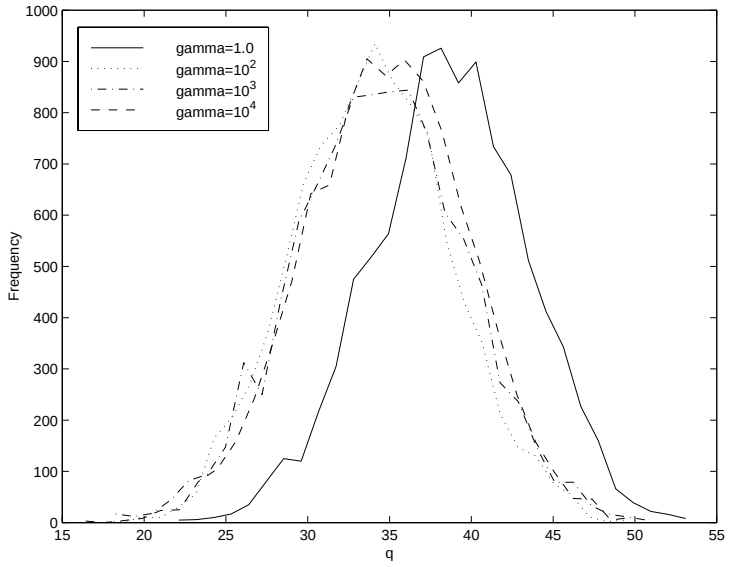

Figure 4.7: Probability distributions for the distance between two points in a sample for the glass (9-1-6 network) experiment.

experiments (as can be seen by examining the mean value in column 5 of Table 4.1).

4.3.3 Discussion

The pairwise distribution $P(q)$ (or second-order statistics) of a weight vector sample can be used to provide information about how certain areas (such as low-lying or AM) are distributed over the error surface. Qualitative differences between $P(q)$ for different MLP error surfaces have been observed. A simple description of this structure is also possible.

The fact that AM often corresponded to only a few error values suggests that they are either very tightly clustered (into a small number of clusters), or that these particular values are present in many places about the error surface. When the pairwise distribution is dominated by distances near zero, the former is true, especially in the 4-1-4, 8-1-8, and 8-2-8 encoders. In the 4-1-4 and 8-1-8 cases, it is impossible for the network to represent the encoding. It is likely that the effect that this has on the error surface means that backpropagation makes little progress from its initial starting point near the origin. However, for many of the other encoders, a wide range of distances between points is shown, meaning AM are scattered over the error surface. A staircase-like error surface is consistent with these observations. Indication of clustering behaviour is also observed in some of the pairwise distributions of low-lying regions of real-world problems.

4.4 Optimization, Landscapes and Surfaces

Optimization problems are found in many different areas of science. In this section the most well-known metaphors associated with these optimization problems are discussed, and some of the techniques which have been used to study the nature of different optimization problems are examined. A method involving correlation statistics and scatterplots is applied to MLP error surfaces.

4.4.1 Fitness Landscapes and Evolutionary Algorithms

In biology, the metaphor of a *fitness landscape* has been widely used since its introduction by S. Wright in the 1930's, to describe the evolutionary process [124]. Each member of a population can be considered a point on a fitness landscape. Biologists have used this metaphor to study

Neural Networks	Evolutionary Algorithms	Optimization and Other Areas
Error Surface	Landscape	Cost, Performance or Response Surface
Weight Vector	Individual	Trial Solution, Point
Error Function	Fitness Function	Cost or Objective Function

Table 4.2: Terminology related to optimization problems.

the effects of different evolutionary operations, speciation and other aspects of the evolutionary process. Moving uphill on the landscape is in a direction of increasing fitness (see, e.g. [76]). With the development and application of evolutionary-inspired algorithms in computer science, the terminology of fitness landscapes has been used in the context of optimization problems. Table 4.2 lists some of the equivalent terminology used in discussing optimization and search problems.

Evolutionary Optimization research has developed the fitness landscape metaphor in an attempt to better understand the nature of problem solving with Evolutionary Algorithms (EA's). Despite the obvious similarities, there is an important difference between this area of research and that of MLP error surfaces. Evolutionary Algorithms and theoretical evolutionary biology have mainly been concerned with discrete problem spaces. A fitness landscape is hence defined as a *discrete* structure, that is "a labeled, directed graph" [118]. Working with this landscape, it is possible to develop precise theory and definitions supporting the structure of the landscape [118, 226], based on each point having a finite neighbourhood of surrounding points on the landscape (e.g. a point is a minimum if its fitness is greater than that of any adjacent vertex). It follows that distance measures between points on the landscape are also discrete.

Discrete EA's have found large application in the domain of combinatorial optimization, where the solution space is also discrete. Some work has also been concerned with the exploration of the structure of the configuration spaces of combinatorial optimization problems. Given a discrete optimization problem, it is always possible (theoretically) to exhaustively evaluate the cost function for all points on the landscape. Statistics such as the number of global and local minima can then be derived.

Discrete Space Results: the Central Limit Catastrophe and Extra-dimensional Bypass

Two results exist that are mentioned here because of their possible relevance to the MLP training optimization problem. Although this work has been developed in the domain of finite discrete

search spaces (i.e. a graph), it seems possible that they may apply to some extent in the continuous case.

It has been suggested that combinatorial optimization problems exhibit a *central limit* or *error catastrophe* as the size of the problem increases. Kauffman suggests that as problem size increases, the fitness value of local minima tends towards the fitness of a random solution (except for some exponentially shrinking fraction of the minima) [124]. Boese suggests that it may actually be that the fitness values of the local minima shrink to the average fitness of a random *local minima* [36]. The central limit theorem may provide a reason for this relationship, the variance of the distribution of local minima shrinking as the size of the problem increases. This phenomenon has yet to be satisfactorily demonstrated or explained in the literature.

Another result which comes from biology is sometimes called the *extra-dimensional bypass* result [2, 53, 54]. It is argued that high-dimensional fitness landscapes or error surfaces are "robustly dominated" by saddle points. To see this result, consider the Hessian at some point on the surface. Construct an associated incidence matrix $\mathbf{A}$, which has elements $+1$ where corresponding elements of $\mathbf{H}$ are positive and -1 values where corresponding elements of $\mathbf{H}$ are negative. The May-Wigner stability theorem [90] is claimed to show that as the dimensionality becomes large, the probability that all eigenvalues of $\mathbf{A}$ are negative (i.e. a local minimum) rapidly tends towards zero.

Although this thesis and other work on MLP error surfaces supports the notion that local minima are rare and saddles or plateaus may be very common, there remains a significant gap in the research towards reconciling the extra-dimensional bypass result with any practical situation.

4.4.2 Correlation Measures of Landscapes

A number of researchers have considered examining the correlation between fitness and distances between points on discrete fitness landscapes. These studies include the cost versus average distance of an optimum point to all other points within a sample of local optima [36, 155], cost versus distance of local optima from the best optima found [124, 125]. These studies on artificial and combinatorial optimization landscapes have indicated that a "Massif Central" [124] or "big valley" [36] structure seems to exist in many landscapes. That is, perhaps not surprisingly, cost in general seems to increase with distance from the best minimum, providing an

intuitive picture of the landscape as a big bowl-like structure with smaller ridges, valleys and other structure imposed on it. These approaches require collecting a sample of local optimum points, which is infeasible in the continuous case.

Random walks on landscapes have been analyzed as time series data [104, 105, 247]. This approach allows a correlation length to be calculated, that is, a measure of the distance between two points at which the value of one point still provides some significant information about the other [104]. An important assumption for this analysis is that the landscape is *statistically isotropic*, meaning that the statistical properties of a random walk are the same regardless of the starting position on the landscape. In this sense an isotropic landscape *"looks (globally) the same everywhere"* (Hordijk [104], p. 7). Given the known results concerning MLP error surfaces, it seems unlikely that this assumption is valid in this case. In addition, there is the problem of choosing a step size on the continuous error surface, and the isotropic assumption will be related to this choice.

One correlation measure which can be adapted to continuous error surfaces is the Fitness Distance Correlation (FDC) of Jones [118]. FDC is intended to provide a measure of the global structure of the surface in question, by examining how the value of the cost (fitness) function varies with the distance between a given point and a global optimum. A sample of random points on the surface is chosen, and the standard sample correlation coefficient is calculated:

$$r = \frac{Cov(D,E)}{\sigma_D \sigma_E} \tag{4.1}$$

$$= \frac{n\sum^n DE - \sum^n D \sum^n E}{\sqrt{[n\sum^n D^2 - \sum^n D^2][n\sum^n E^2 - \sum^n E^2]}} \tag{4.2}$$

where $Cov(D,E)$ is the covariance of D and E, D is a set of distances for a sample of points, E is the corresponding set of fitness function values for the sample and σ_X is the standard deviation of X.

FDC was proposed as an indicator of the difficulty of a search space for a Genetic Algorithm (GA) [80] to solve - the implication being that

> *. . . it is the relationship between fitness and distance to the goal that is important for GA search.* (Jones [118], p. 134)

The results support this measure, indicating that problems with a low FDC coefficient are hard for a GA to solve, whereas those with a high FDC coefficient value are GA-easy. While it has been shown that this correlation coefficient is not always successful at predicting the "GA-difficulty" of a problem [1], Jones suggests that FDC scatterplots provide a useful visualization of this relationship, even when the FDC coefficient does not summarize the relationship well.

One important prerequisite for calculating the FDC is knowledge of the global minimum. Although this knowledge is not normally the case for MLP error surfaces, the student-teacher model (see Section 3.6.1) provides knowledge of a global minimum through the creation of a randomly generated teacher network.

In the following experiments, the student-teacher model is used to generate FDC results for MLP's. Throughout, MSE is used as the error function and Euclidean distance is the distance measure. Teacher networks were generated by choosing their weights from a $\mathcal{N}(0, 5)$ distribution, in an attempt to generate networks with realistic weights (i.e. some large weights leading to some units saturating their outputs). The "student" networks' weights were chosen from a $\mathcal{N}(\mu_{teacher}, 1)$ distribution, and the weight vector was then scaled to a length chosen from $\mathcal{U}[0, 100]$. 5000 points were used to calculate the FDC coefficients, while only 2000 were used for the fitness-distance scatterplots for clarity. The number of input units, number of hidden units and the number of training patterns were varied in the experiments. All networks had a single output unit. The hidden and output units used the $tanh$ activation function. Each unique training instance was run 10 times with different random initializations of teacher networks.

The FDC coefficient results are shown in Table 4.3. Each table entry reports the mean and standard deviation of the 10 different experiments for each training instance. Firstly, all coefficient values are positive, confirming intuition that moving away from a global minimum cannot lead to a *decrease* in error. At worst ($r \approx 0$), there is basically no correlation between the error value and the distance to the global minimum - the current location of a search or the trajectory up to some point in the search yields no information regarding the location of the global minimum. A general trend in this table is from relatively low r values for small numbers of inputs, hidden units and training patterns, to high values as these variables are increased. Standard deviations can be reasonably high for small values of the variables, and remain high even for large training sets when the network size is small.

From these r values alone, the indication is that for networks with more inputs and hidden

No. Patterns	No. Inputs (y)/No. Hidden (x)				
		1	5	10	100
1	1	0.185 (0.069)	0.214 (0.074)	0.228 (0.101)	0.269 (0.097)
	5	0.245 (0.067)	0.189 (0.104)	0.273 (0.066)	0.248 (0.100)
	10	0.207 (0.088)	0.260 (0.064)	0.249 (0.094)	0.198 (0.130)
	100	0.322 (0.083)	0.283 (0.049)	0.183 (0.123)	0.113 (0.096)
5	1	0.176 (0.073)	0.285 (0.083)	0.378 (0.049)	0.423 (0.073)
	5	0.266 (0.078)	0.401 (0.075)	0.409 (0.400)	0.466 (0.043)
	10	0.304 (0.082)	0.466 (0.053)	0.493 (0.051)	0.480 (0.087)
	100	0.403 (0.091)	0.476 (0.089)	0.494 (0.137)	0.309 (0.148)
10	1	0.213 (0.069)	0.318 (0.073)	0.314 (0.059)	0.459 (0.059)
	5	0.305 (0.058)	0.469 (0.073)	0.494 (0.075)	0.588 (0.107)
	10	0.317 (0.076)	0.529 (0.054)	0.573 (0.086)	0.594 (0.067)
	100	0.454 (0.128)	0.548 (0.103)	0.612 (0.063)	0.523 (0.089)
100	1	0.194 (0.073)	0.320 (0.120)	0.401 (0.056)	0.529 (0.051)
	5	0.378 (0.121)	0.534 (0.072)	0.689 (0.064)	0.827 (0.037)
	10	0.383 (0.140)	0.670 (0.085)	0.744 (0.060)	0.880 (0.020)
	100	0.418 (0.108)	0.787 (0.117)	0.883 (0.056)	0.864 (0.026)
1000	1	0.180 (0.069)	0.280 (0.104)	0.382 (0.065)	0.488 (0.063)
	5	0.335 (0.152)	0.585 (0.059)	0.694 (0.052)	0.845 (0.024)
	10	0.377 (0.110)	0.631 (0.122)	0.798 (0.055)	0.926 (0.017)
	100	0.506 (0.161)	0.854 (0.050)	0.926 (0.044)	0.970 (0.005)

Table 4.3: Fitness-Distance correlation values for student-teacher experiments.

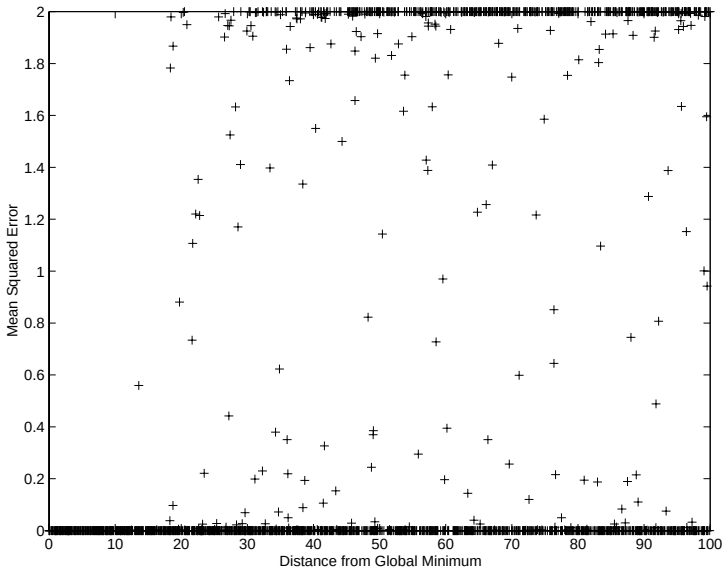

Figure 4.8: FDC scatterplot; 10-5-1(#1) network; $r = 0.3846$.

units, distance from the global minimum and the current error value are related - implying that algorithms that are able to utilize this property will perform well in this situation. To allow a more detailed insight into the nature of these error surfaces however, the scatterplots of the experiments must be examined.

Figures 4.8-4.15 show a representative sample of the kinds of scatterplots which were observed from the experiments. The general nature of all of the experiments can be summarized by this sample.

Figure 4.8 shows an FDC scatterplot for a 10-5-1(#1) network (i.e. a 10-5-1 MLP with a single training pattern). Clearly, two error values dominate the sample, over the range of distances examined - one which is very close to the global minimum and one which is also the highest (worst) error value found ($E \approx 2$). This result indicates that this error surface is largely dominated by two plateaus, and the small number of points at intermediate error values suggests that the transition between these two levels is relatively sharp. Note also that the high plateau is not seen until a distance of roughly $||\mathbf{w}|| \simeq 20$ from the global minimum. In this case, the area surrounding the global minimum is quite flat, which will cause problems for algorithms seeking to converge to the exact global minimum. In practice however, this may be less of a concern - any point with an error so close to zero would probably be sufficient to halt training.

In Figure 4.9 the number of training patterns has increased to 5 for a 10-5-1(#5) network. The observable effect is the appearance of a number of intermediate levels between the $E \approx 0$ and the $E \approx 2$ levels. In addition, the levels appear incrementally as the distance increases. This

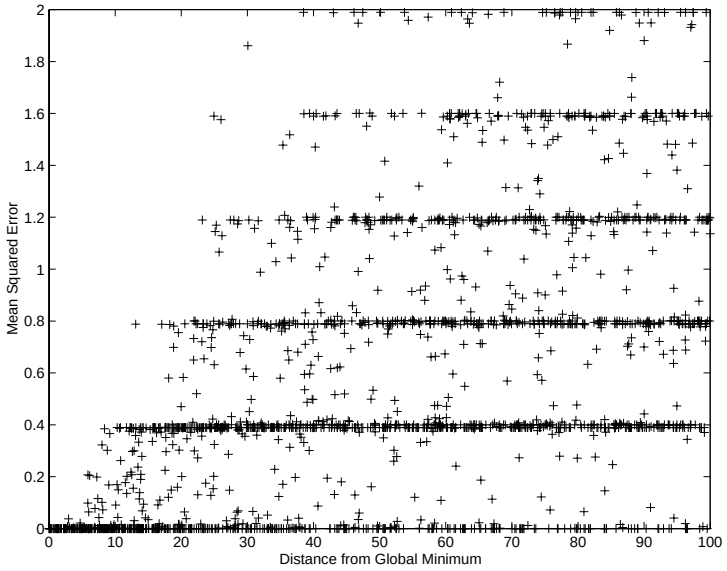

Figure 4.9: FDC scatterplot; 10-5-1(#5) network; $r = 0.5235$.

observation accounts for the higher correlation coefficient ($r = 0.5235$) compared to Figure 4.8. The levels are equidistant, separated by $E \approx 0.4$ in error. This result implies that the number of levels are directly related to the number of training patterns in this experiment. Lastly, the number of intermediate points between the levels has increased. The overall impression is that the global minimum is now situated in a "smaller bowl", in that walking a shorter distance from the global minimum leads to much worse error values. The number of points at a level close to the global minimum also drops off with increasing distance, again suggesting a unique global minimum and surrounding neighbourhood.

Increasing the number of training patterns further leads to an extension of these trends. Figure 4.10 shows the scatterplot for a 10-5-1(#10) network. A greater number of levels dominate the plot, though the levels themselves are becoming less prominent as the number of points between them increases and as they become closer together. The rate at which higher error values appear moving away from the global minimum has also increased, indicating a reduced region around the global minimum with small error values (and producing a lower correlation value $r = 0.4330$. At a large distance, points with $E \approx 0$ are becoming increasingly rare.

The next scatterplot is a 1-1-1(#1000) net, shown in Figure 4.11. This small network with large training set produces a plot dominated by two levels similar to Figure 4.8. In this case however, the number of intermediate points has increased substantially, and as distance increases the number and magnitude of points with worse error values increases more rapidly (leading to a low $r = 0.1501$).

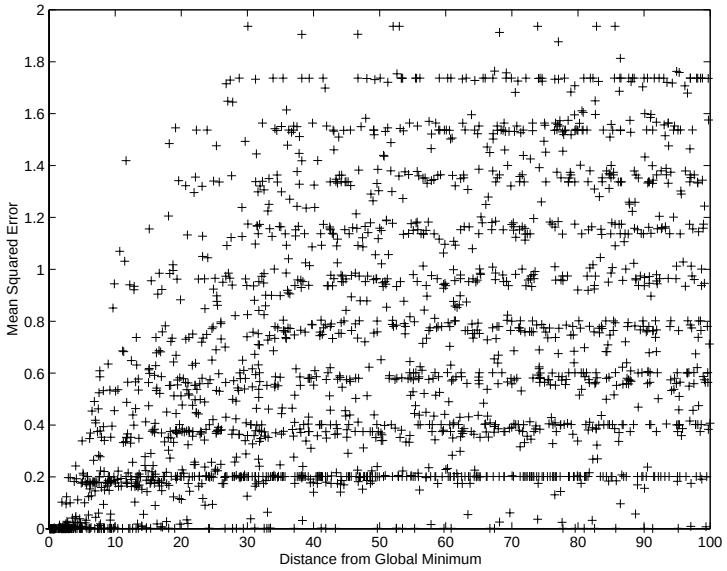

Figure 4.10: FDC scatterplot; 10-5-1(#10) network; $r = 0.4330$.

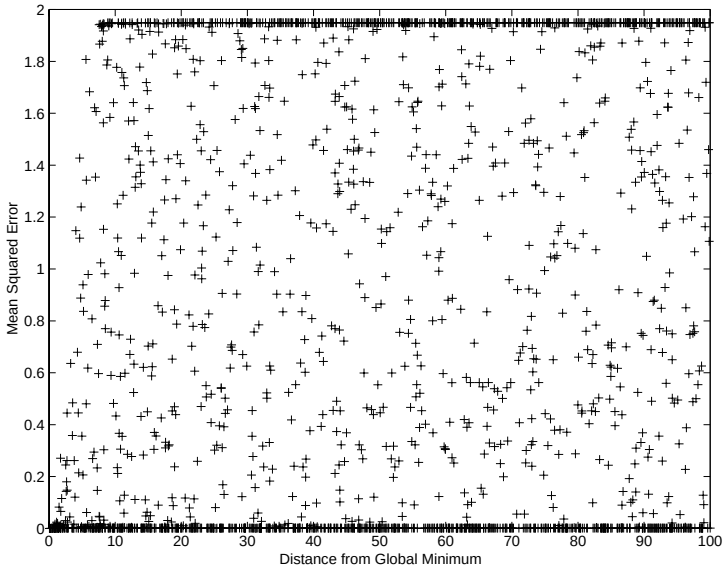

Figure 4.11: FDC scatterplot; 1-1-1(#1000) network; $r = 0.1501$.

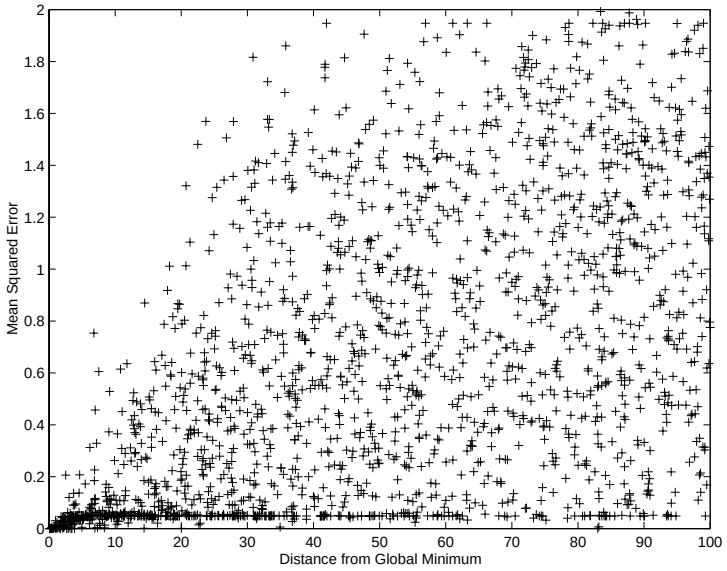

Figure 4.12: FDC scatterplot; 1-100-1(#1000) network; $r = 0.5040$.

In Figure 4.12 (1-100-1(#1000) net), the levels which have dominated the above examples are all but gone. A significant number of points exist at one level at a value slightly above the global minimum, but only for smaller distances. A wide range of different error values is spread over the majority of the distance values. Note however that the intuition of a bowl surrounding the global minimum is supported with the smooth increase in worse error values moving away from the global minimum. The value of r has increased significantly from Figure 4.11 to Figure 4.12, due mainly to the disappearance of the high plateau of points and the diminishing lower plateau for high values of distance from the global minimum.

A scatterplot produced by a 100-1-1(# 1000) experiment (Figure 4.13) shows some similarity to Figure 4.11. Two levels of error dominate the scatterplot, but the level close to zero error is much more common, and appears even at large distances from the global minimum. A greater number of points appear between these levels than in Figure 4.11. Also, the higher error level has dropped to a smaller value ($E \approx 1.6$) than in previous figures.

Several of the experiments with larger networks (with moderate to large training sets) produced a scatterplot similar to that shown in Figure 4.14 (100-100-1(# 1000) network). Now the picture is of a unique global minimum, about which error steadily rises. The error values are concentrated into a tube, reinforcing the idea of a big-valley structure and suggesting an overall smoother surface where a multitude of different error values can be found within this tube. At larger distances from the global minimum, a leveled-structure appears, and error values become concentrated around these intermediate error levels.

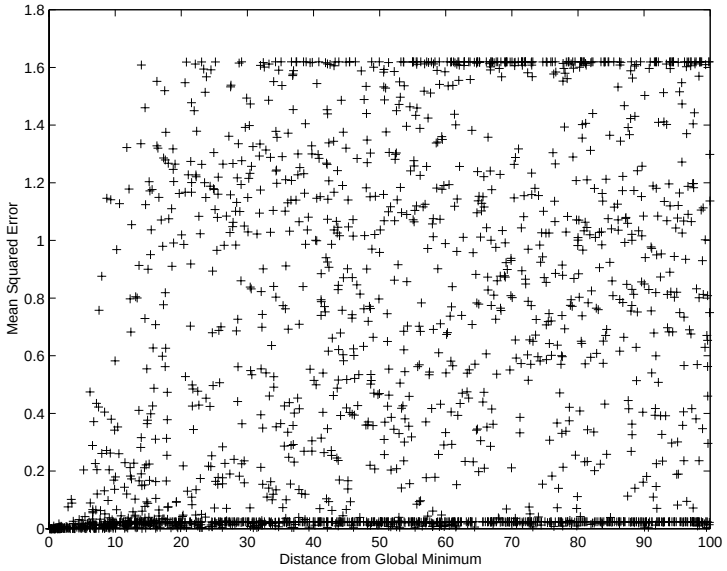

Figure 4.13: FDC scatterplot; 100-1-1(#1000) network; $r = 0.3064$.

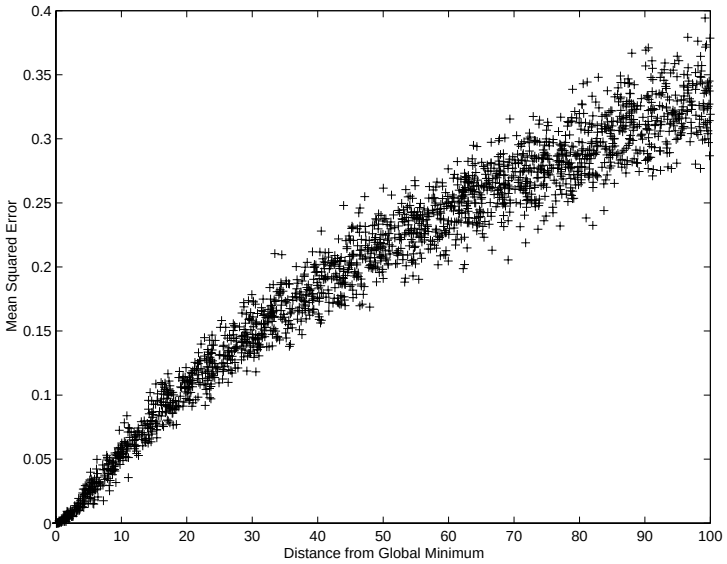

Figure 4.14: FDC scatterplot; 100-100-1000(#1000) network; $r = 0.9706$.

4.4.3 Discussion

These results show that a number of different phenomena can be observed in the structure of MLP error surfaces. A small number of training patterns induces a number of distinct levels onto the error surface. For small values of inputs and hidden units, the number of levels was observed to have an upper bound of $(k + 1)$, where k is the number of training patterns. As the number of training patterns increases, the effect becomes more like a smoothing of the error surface - as levels blur together and intersect in various different ways. Restricted numbers of hidden units and inputs (e.g. Figures 4.11 and 4.13) seem to be factors in producing levels in the scatterplots. As all three variables (inputs, hidden units, number of training patterns) become large, the error surface becomes globally smoother, with a near-continuous range of error values at various distances from the global minimum.

The above experiments attempt to describe the fundamentally different features of scatterplots observed. Nevertheless, several experiments produced plots whose behaviour can be described as some kind of combination of such features. An example is shown in Figure 4.15, for a 100-1-1(# 100) experiment. The "tube"-like structure seen in Figure 4.14 is evident for lower values of distance and error, but a two-leveled structure becomes increasingly prominent as distance increases (cf. Figure 4.13). Three of the ten 100-1-1(# 100) experiments produced scatterplots similar to Figure 4.15, while the other seven closely resembled Figure 4.13. A more detailed study would be required to explain the interaction of these effects with varying numbers of inputs, hidden units and numbers of training patterns.

Overall, the FDC experiments indicate that the error surface has a complex structure. Furthermore, the structure of the error surface is sensitive to factors that are commonly varied when MLP's are used - the number of inputs, number of hidden units and the number of training patterns. This observation suggests that the performance of a learning algorithm can be expected to vary widely across different values for these variables, and across different training tasks. The positive values in Table 4.3 support the idea of a global big valley structure in these MLP error surfaces. However, this picture is an oversimplification, as can be seen by the variety of structure in the FDC scatterplots above.

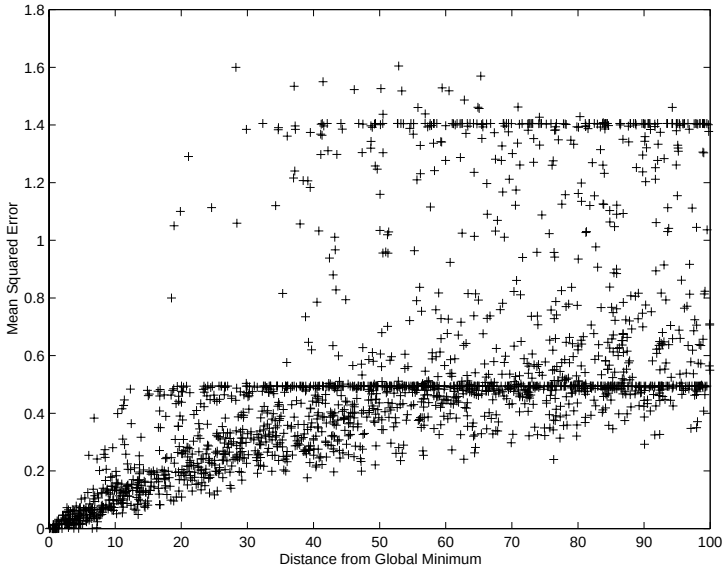

Figure 4.15: FDC scatterplot; 100-1-1(#1000) network; $r = 0.5748$.

4.5 Statistical Physics, MLP's and Combinatorial Optimization Cost Surfaces

The field of statistical physics has overlapped with ANN research in a number of different ways. Techniques in statistical physics that describe the behaviour of macro-systems in terms of their large number of constituent micro-system parts have been applied to neural networks as being large systems of simple processing elements which interact through the network connectivity. Statistical physics was also the origin for work into the analysis of the search spaces of combinatorial optimization problems. The development of simulated annealing in particular led to the consideration of the application of the theory of spin glasses (magnetic systems modeled as a lattice with an "atomic magnet" (e.g. spin up or down) at each lattice point) to combinatorial optimization [36]. In this section a number results from this domain which are particularly related to the MLP training problem are discussed.

4.5.1 Chaos, Fractals and the Error Surface

Several studies in the literature have explored the possible connections between MLP's, fractals and chaos theory. Choie et al. [45] study the output values of an MLP trained on the XOR problem as a function of backpropagation learning epochs. As the learning rate η is varied, chaotic oscillations and period doubling bifurcations are observed in the output values. These

results are obtained for large numbers of hidden nodes (typically 300), but the behaviour is claimed to exist for more practical (e.g. two or three) hidden units. The authors attempt to relate these results to the assumption of ravine-like structure in the error surface.

Taking a simpler model, Rojas [197, 198] has shown that for a single linear neuron unit, the sequence of on-line backpropagation weight updates approximates the iterated function system of the attractor defined by the input patterns; that is, the iteration path is fractal. Batch backpropagation with a momentum term is also examined. The 2-D space defined by η and the momentum parameter is shown to consist of two regions of convergence and divergence for a linear unit with a single input trained using batch backpropagation. Configurations on the boundary between these regions lead to stable oscillations in the iteration process. Optimal combinations of the two parameters (in the sense of fastest convergence to some criterion) are found to lie on a line which also has a fractal structure.

Kahng claims to have found evidence that random walks over high-dimensional MLP error surfaces are statistically fractal [121]. Specifically, analysis of the spectral properties of a random walk allow parameterization in terms of a fractional Brownian motion model. It is suggested that this property could be tested for efficiently, preceding a simulated annealing algorithm which may be efficient on such a landscape (see next section). Unfortunately, the claims are not supported by evidence, and the practical utility of such an approach has not been demonstrated.

4.5.2 Ultrametricity

One structural metric property that has emerged from analysis of spin-glasses is known as *ultrametricity* (see [188] for a review). Ultrametricity is a concept that originated from mathematical topology in relation to p-adic numbers in the mid 1940's. It was applied in the field of taxonomy in the 1960's, being a natural metric for the description of points on a hierarchical tree structure. The discovery of ultrametricity in disordered systems such as spin glasses led to the examination of the configuration spaces of combinatorial optimization problems for similar structure. These include traveling salesman [129], circuit placement [221] and graph colouring [10, 38] problems.

In general, a distance in a metric space obeys the following conditions

$$d(\mathbf{w_a}, \mathbf{w_b}) = 0 \text{ iff } \mathbf{w_a} = \mathbf{w_b}$$

$$d(\mathbf{w_a}, \mathbf{w_b}) = d(\mathbf{w_b}, \mathbf{w_a})$$

$$d(\mathbf{w_a}, \mathbf{w_c}) \leq d(\mathbf{w_a}, \mathbf{w_b}) + d(\mathbf{w_b}, \mathbf{w_c}).$$

The last of these conditions is the standard triangle inequality. An ultrametric space is endowed with an (ultrametric) distance measure satisfying a stronger version of the triangle inequality

$$d(\mathbf{w_a}, \mathbf{w_c}) \leq Max\{d(\mathbf{w_a}, \mathbf{w_b}), d(\mathbf{w_b}, \mathbf{w_c})\}.$$

For any three points in such a space, the two of them that are nearer to each other are equidistant from the third. This condition means that all triangles in an ultrametric space must be either equilateral, or isosceles with a small base (i.e. third side shorter than the two equal ones). Points in an ultrametric space can be interpreted as a tree, with the distance between any two points in the tree defined by the height[7] of their first common ancestor. This strict definition is used as a reference for the analysis of configuration spaces and other systems, where a set of points is examined for its *degree of ultrametricity*. As an example, in an N-dimensional Euclidean space, a maximum of $N + 1$ points can satisfy the ultrametric inequality. Randomly selected points in this case can be expected to display a low degree of ultrametricity [188].

Ultrametric points follow a recursive or hierarchical structure and consequently are fractal. In [223] Sorkin proves that for an optimization problem with a perfectly fractal cost or energy landscape, simulated annealing can be an efficient search strategy, that is the expected energy of a solution ϵ can be found in time polynomial in $1/\epsilon$, where the exponent of the polynomial depends on properties of the fractal. This result scales up to higher dimensions. The interest in ultrametric structure is the suggestion that algorithms might be developed for certain practical optimization problems which run in polynomial time.

[7] Meaning distance from the root of the tree

It is easy to define a distance measure on an energy landscape which induces an ultrametric set. The minimax energy, defined as the lowest maxima of any path connecting two points $\mathbf{w_a}$ and $\mathbf{w_b}$, is an ultrametric measure [145]. Detection of ultrametric structure in some metric space thus implies a correlation between that given distance measure (e.g. Euclidean distance) and the minimax energy. In this case, minima which are close to each other are separated by small energy barriers, whereas distant minima are separated by large energy barriers. Such a landscape is said to be *quasi-fractal* [145]. It has been suggested that simulated annealing can also work well on such quasi-fractal landscapes [221].

4.5.3 Detecting Ultrametricity in MLP Error Surfaces

Given a sample of points in a configuration space (such as MLP weight space), the degree of ultrametricity can be measured using correlation functions of distances between sample points in the configuration space. Such techniques were originally developed and applied to discrete configuration spaces.

The joint probability $P(q_1, q_2)$ can be defined as the probability of selecting points $\mathbf{w}_a$ and $\mathbf{w}_b$ at a distance q_1 and q_2 apart from a third point $\mathbf{w}_c$. Define the following correlation function [129]

$$C_1(q_1, q_2) = P(q_1, q_2) - P(q_1)P(q_2)$$

where $P(q_1, q_2)$ is the probability that a triangle will have *any* two sides of length q_1 and q_2. $C_1(q_1, q_2)$ is positive for a heterogeneous sample of distances and indicates possible ultrametric structure through peaking along the diagonal $q_1 = q_2$ (though ultrametric sets may contain off-diagonal peaking when using this function). However, $C_1(q_1, q_2)$ is also positive for isosceles triangles with a long base. If the sides of triangles are labeled such that $q_1, q_2 \geq q_3$, a second correlation function [221]

$$C_2(q_1, q_2) = \widetilde{P}(q_1, q_2) - \widetilde{P}(q_1)\widetilde{P}(q_2),$$

can be used, where $\widetilde{P}(q_1, q_2)$ is the probability that a triangle will have its *two longest sides* of length q_1 and q_2, and $\widetilde{P}(q)$ is the probability that one of the two longest sides will have length q. $C_2(q_1, q_2)$ tests specifically for ultrametric structure, but will contain peaks for random

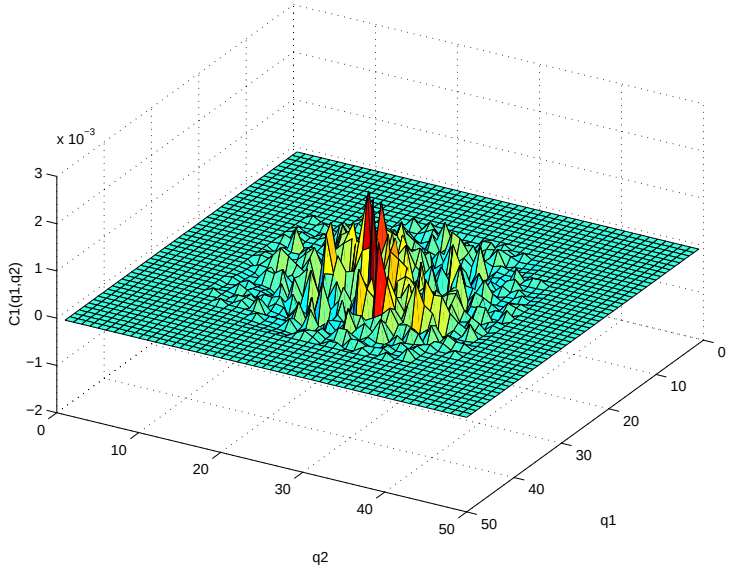

Figure 4.16: C1 statistics for triangles formed by random ($\gamma = 1.0$) samples on the error surface for the cancer (9-1-2 network) experiment.

(homogeneous) samples of distances. Hence $C_1(q_1, q_2)$ and $C_2(q_1, q_2)$ provide complementary information.

To compare the degree of ultrametricity between different configuration spaces, the following correlation coefficient may be used [221]

$$s = \frac{\langle (q_1 - \langle q_1 \rangle)(q_2 - \langle q_2 \rangle) \rangle}{\sigma_1 \sigma_2}$$

where $\langle \dots \rangle$ indicates averaging over $\widetilde{P}(q_1, q_2)$ and σ_i is the standard deviation

$$\sigma_i = \langle (q_i - \langle q_i \rangle)^2 \rangle^{\frac{1}{2}}.$$

4.5.4 Results: Samples of Low-lying Regions

Experiments were conducted on samples of low-lying regions of MLP error surfaces to test the degree of ultrametric structure present. The cancer, credit card and diabetes datasets were used as in Section 4.3.2 above.

Given a sample of low-lying points on the error surface, distributions for $C_1(q_1, q_2)$ and $C_2(q_1, q_2)$ were produced by generating groups of three points selected uniformly from the

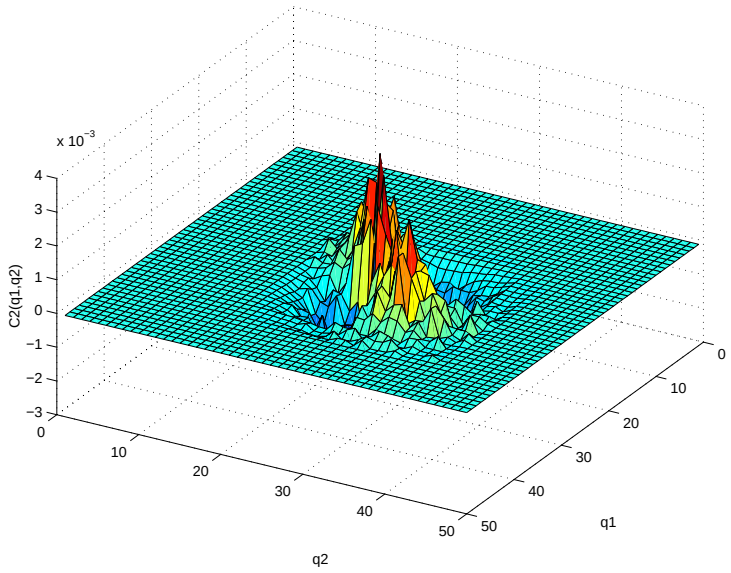

Figure 4.17: C2 statistics for triangles formed by random ($\gamma = 1.0$) samples on the error surface for the cancer (9-1-2 network) experiment.

low-lying points. These "random triangles" generate third-order statistics of low-lying areas and can be used to test for degree of ultrametric structure via these correlation histograms. Figures 4.16 and 4.17 show $C_1(q_1, q_2)$ and $C_2(q_1, q_2)$ for the cancer (9-1-2 network) experiment with $\gamma = 1.0$. The small peak evident in Figure 4.16 and the noisy distribution of Figure 4.17 give an indication of the kind of results that these tests return on points selected randomly from a high-dimensional space. Greater peaking on these distributions gives evidence for a sample to be containing a level of ultrametric structure greater than this random level. The correlation coefficients s for experiments using the cancer, credit card and diabetes datasets are shown in Column 6 of Table 4.1. Note that for many of the experiments the values of s obtained for low-lying regions ($\gamma < 1.0$) are not significantly different from the s values for $\gamma = 1.0$ (i.e. uniform random sampling of the error surface). These experiments produced similar sample distributions as those shown in Figure 4.16 and Figure 4.17, with little observable differences as γ was decreased.

For these experiments, ultrametric structure greater than random levels was detected predominantly in networks with 1 hidden unit. The cancer (9-1-2 network) experiment indicated a high level of ultrametricity for low-lying solutions, whilst the glass (9-1-6 network) and diabetes (8-1-2 network) experiments indicate moderate ultrametric structure. For these experiments, increasing γ increases the level of ultrametricity (see Table 1), whilst for the larger networks γ has little influence. The $C_1(q_1, q_2)$ and $C_2(q_1, q_2)$ distributions for the cancer (9-1-2 network) are

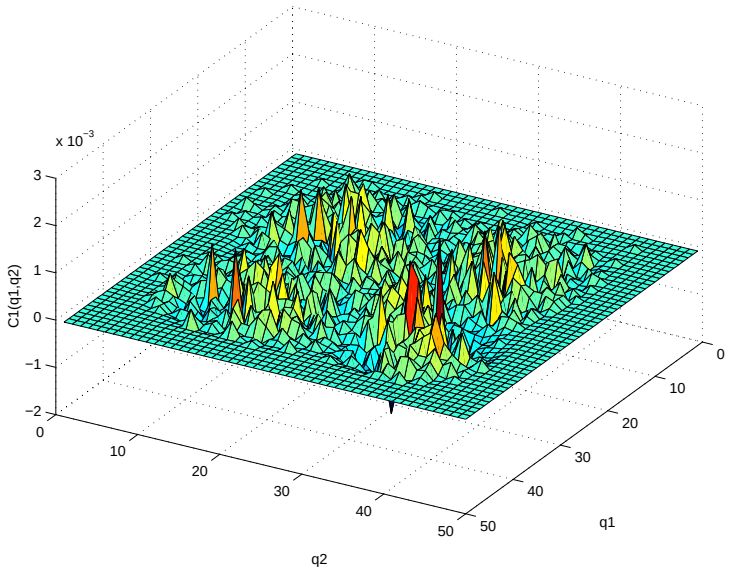

Figure 4.18: C1 statistics for triangles formed by random ($\gamma \approx 10^4$) samples on the error surface for the cancer (9-1-2 network) experiment.

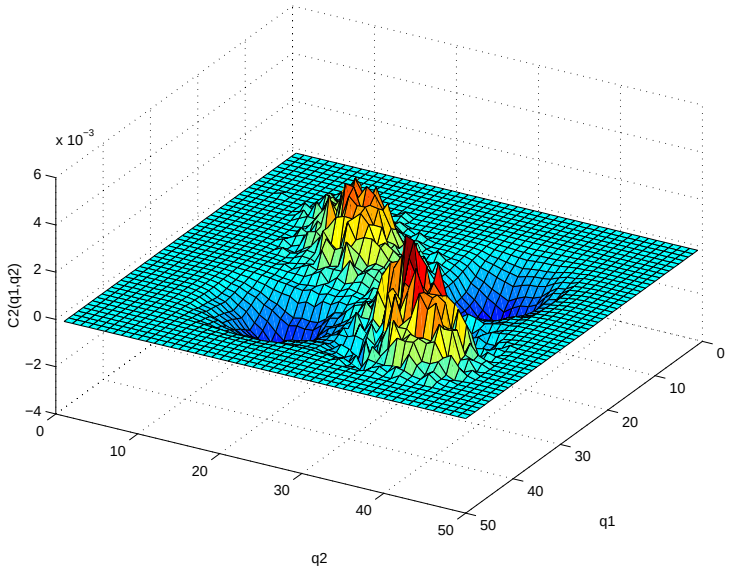

Figure 4.19: C2 statistics for triangles formed by random ($\gamma \approx 10^4$) samples on the error surface for the cancer (9-1-2 network) experiment.

shown in Figures 4.18 and 4.19 respectively. A significant amount of peaking is evident in these figures, in comparison to Figure 4.16 and Figure 4.17. This observation is in agreement with the s values for this experiment (column 6 of Table 4.1). A recurring trend in the s values for the experiments is a "jump" as the data moves from random ($\gamma = 1.0$) to low-lying ($\gamma < 1.0$) samples.

The bimodal nature of the low-lying data for the cancer (9-1-2 network) experiment is also reflected in Figures 4.18 and 4.19. Note that $C_1(q_1, q_2)$ contains clusters of bumps on either side of the two bumps lying along the $q_1 = q_2$ diagonal. These clusters correspond to occurrence of (non-ultrametric) isosceles triangles with a long base.

4.5.5 Results: Samples of Apparent Minima

The samples of AM from the encoder network problems were also examined for degree of ultrametricity. An alternative definition of the correlation coefficient s used above is the distribution-free rank correlation coefficient s_r

$$s_r = 1 - \frac{6 \sum_{i=1}^{k} d_i^2}{k(k^2 - 1)}$$

where k denotes the number of points in the sample, and d_i is the difference between the ranks of the ith pair of longest sides. This statistic has an advantage when the distribution of the samples is unknown, or is significantly non-normal. This advantage must be traded-off with an increased computation time.

Figures 4.20 and 4.21 show values of s_r for several of the AM samples. To obtain an indication of the dynamics of s_r, calculation of s_r values was repeated several times during the training processes which produce the AM. These calculations give an idea of how the learning trajectories become ultrametrically distributed as a function of the number of epochs. While the random starting points return a small value of s_r, it is observed that as training progresses the distribution of the points on the error surface becomes highly ultrametric.

Although it is difficult to conceptualize what ultrametric structure might look like on a high-dimensional error surfaces, some attempt can be made to visualize this structure, using PCA. Figure 4.22 shows an example of such a plot for the apparent minima of a 4-1-4 network, with points collected after 30000 epochs ($s_r = 0.93$). The first three PC's of the AM data for

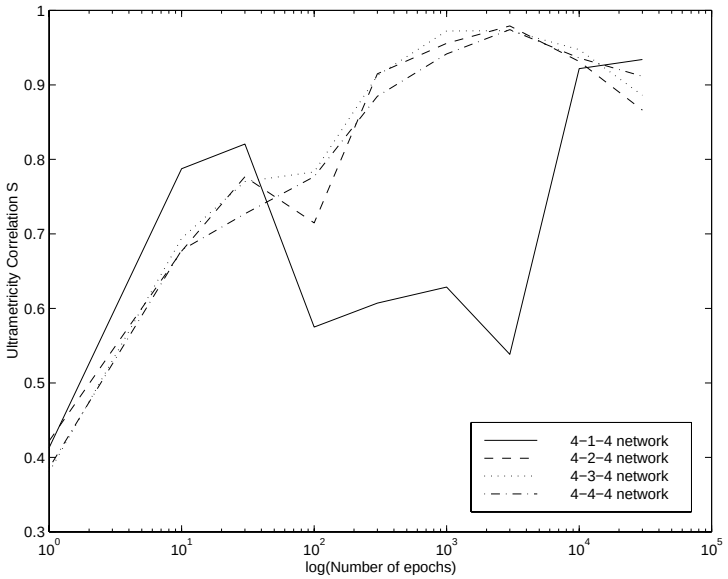

Figure 4.20: Degree of ultrametricity in samples of AM as a function of the number of epochs.

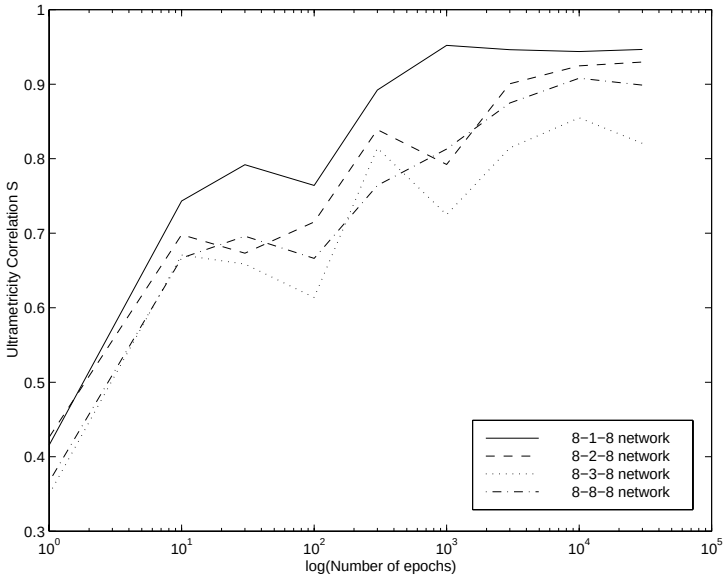

Figure 4.21: Degree of ultrametricity in samples of AM as a function of the number of epochs.

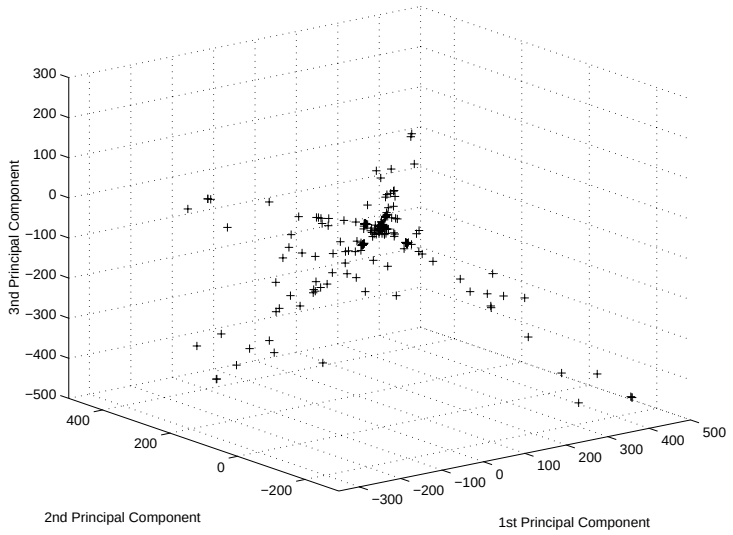

Figure 4.22: A visualization of how the AM are organized in weight space, using the first three principal components, for the 4-1-4 encoder.

this experiment are plotted. In this example $\approx$ 83% of the distance information on the error surface is captured by the first three principal components. It is clear that the structure visible in Figure 4.22 is complex and occurs over a wide range of distances in weight space. The density of the points towards the centre of the figure bear some resemblance to a fractal or hierarchical structure.

4.5.6 Discussion

These experiments show that a significant amount of ultrametric structure can be found both in samples of the low-lying areas and apparent minima of MLP error surfaces. This structure is quite unexpected. Previously [188, 221], the configuration spaces that have displayed this structure were defined on discrete spaces (together with an appropriate distance measure), and are thus quite different to the Euclidean setting considered here. No relationship is evident between the size of the network, in particular the number of hidden units, and the degree of ultrametricity. Other factors such as the dataset are of course involved in the properties of the error surface. The permutation and sign-flip symmetries were removed from the AM samples, thus ultrametricity represents a further level of structure in the error surface.

The temporal nature of the degree of ultrametricity has been explored for samples of AM. It is shown that the degree of ultrametricity increases rapidly from the start of training runs with the number of epochs. This result indicates that the trajectories followed by smooth gradient de-

scent rapidly become ultrametrically distributed, not just the AM obtained after a large number of epochs.

4.6 Summary

This chapter has discussed the error surface metaphor in the training of MLP's. A variety of methods for the exploration of the error surface have been developed and/or applied, using stochastic sampling techniques and methods from analysis of other complex parameterized systems. These methods require no assumptions on the error surface or configuration space under consideration, and have been developed from work on discrete configuration space analysis. In addition, obtaining structural information by sampling methods is simple to implement and is computationally efficient for high-dimensional problems such as MLP error surfaces. These methods can be used to compare the difficulty of different learning problems for an MLP, and in assessing the performance of local and global optimization algorithms for training. Although a general precise description of the error surface is not possible, the methodology makes it possible to describe the global features of the error surface, such as distributions of error values and magnitude of the slope of the surface.

A high degree of ultrametricity, a hierarchical, quasi-fractal structural property, has been discovered to occur in samples of low-lying points and apparent minima of several MLP error surfaces. Previous work has identified ultrametric structure in other problem configuration spaces, but these results have been for discrete spaces. The results of this chapter make a connection between these different problems. The presence of ultrametric structure in the error surface suggests that algorithms which can detect this structure and effectively negotiate it (as simulated annealing does for a purely ultrametric configuration space) will achieve improved optimization performance.

Chapter 5

Probabilistic Modelling of the Error Surface and Optimization

In this chapter the focus of the thesis is shifted slightly, from visualization and understanding the properties of the error surface, towards methods which combine modelling of the error surface with optimization. A class of optimization algorithms is considered that explicitly constructs a model of the error surface and uses this model to search for a minimum of the surface. The idea is derived from the Population-Based Incremental Learning (PBIL) algorithm [16], but is of broader general interest. The model is adapted on the basis of points sampled from the search space, as well as any other prior knowledge or additional information that may be available to improve the search. It is shown that this idea of modelling the search space can be stated in general terms, as a stochastic optimization algorithm with particular sampling properties. An algorithmic framework for this class of optimization methods is developed. A number of algorithms such as PBIL, various Evolutionary Strategies (ES) and random search methods can be viewed as limited instantiations of this general framework. Viewing algorithms in this light suggests the possibility of adapting a large body of ideas and techniques from unsupervised learning and probability density estimation, to continuous optimization problems such as training MLP's. An additional benefit of the framework is that it provides opportunities for the visualization of optimization algorithms and the error surface itself.

Step 1: Initialize population
$(D^{(k)} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\})$
Step 2: Evaluate fitness of all members of population
$F^{(k)} = \{E(\mathbf{x}_1), \ldots, E(\mathbf{x}_k)\}$
Step 3: Selection of parents from the current population
Step 4: Recombination
Step 5: Mutation
Step 6: Evaluate offspring
Step 7: Select survivors
Step 8: Goto Step 2 (until termination condition)

Table 5.1: Pseudo-code for an EA.

5.1 Evolutionary Optimization Algorithms

The term *Evolutionary Algorithms* (EA's) refers to a broad class of optimization algorithms which take some inspiration from evolutionary or adaptive systems in the biological and physical world and attempt to use these insights to solve problems. This area of research was developed independently by several different researchers from the late 1950's (see [76] for an outline and references), and has become a very large and active field.

EA's applied to optimization tasks operate through a stochastic search process that is inherently parallel. A population of candidate solutions is created within the search space, and evolved according to a number of search operators, such as mutation, selection and recombination. By using the cost function values at each evaluated point, the population tends towards an optimum point of the search space. Pseudo-code for a typical EA is shown in Table 5.1. Some of the principles of evolutionary optimization algorithms can be found in the domain of global optimization [237]. For example, in 1975 Jarvis [114] described an algorithm which considers a number (N) of random search processes that operate in parallel. An N-dimensional probability vector is used to determine which search process will be active (i.e. generate a new search point). The search processes compete for selection via this probability vector, which is updated according to a linear reinforcement update rule. This algorithm bears strong similarities with EA's and in particular Population-Based Incremental Learning, which is discussed further below.

EA's normally only use cost function evaluations (i.e. gradient or higher-derivative information is not required or used). This simplified requirement is an advantage in many practical

situations where such information is not available (e.g. Multi-Layer Perceptrons (MLP's) with non-differentiable activation functions) or is computationally expensive to obtain. Related to this requirement is the fact that EA's are not trapped by local minima and can be shown to converge (in probability) to the global minimum in the same sense as simpler stochastic algorithms that share this property [219] (see also Section 2.6.1).

Despite these properties, such work did not have a significant impact on optimization research of the 1970's and early 1980's, partly because of the limited computational resources of the time for solving high-dimensional optimization problems [28, 155], as well as the relative obscurity of some of the work (see, e.g. the references in [237]) and the domination of the optimization field by its mathematical origins.

This situation has changed in recent years, with a surge of research and significant expansion in the scope of EA's. These include nature-inspired optimization methods such as cultural algorithms [191], mimetic algorithms [174], ant colony optimization [68] and particle swarm optimization [126]. The Population-Based Incremental Learning (PBIL) algorithm and its variants share some common themes with the framework of this chapter and are discussed further below. A successful international conference is largely devoted to this area of research [170], in addition to the overlap with more "traditional" EA and Artificial Life literature.

The nature of the development of EA's has created a number of sub-groups of EA's [52]. In this section a brief summary of the major sub-groups that are directly relevant to optimization is given, and highlight the differences between these sub-groups. Note however that in the last decade or so a great deal of consolidation has occurred in the EA community, so these distinctions are certainly blurred and perhaps somewhat irrelevant, but this change has been of general benefit to the field.

5.1.1 Genetic Algorithms

Genetic Algorithms (GA's) [80, 103] are the most well-known family of EA's. Table 5.1 also serves as an adequate general description of a GA.

The factors that are often said to differentiate GA's from other EA's are:

- Each candidate solution or population member is encoded as a string of representative tokens which is analogous to the biological chromosome. The most common example is to encode solutions as bitstrings.

- The normal operators used are selection, recombination (or crossover) and mutation. These operators are inspired by biological observations.

- Mutation is a very rare event in biological genetics, so recombination is often considered to be the chief operator in the algorithm.

GA's have found application in many real-world optimization tasks which require the solution of a high-dimensional problem. However, GA's were originally developed as more general models of robust, adaptive systems [103]. The fact that solving optimization problems is only one possible application of GA's has been pointed out in the literature [61]. Nevertheless, it remains by far the most widely studied and developed area of applied GA research.

5.1.2 Evolutionary Strategies and Evolutionary Programming

A different group of EA's was pioneered by I. Rechenberg and H-P. Schwefel in Germany in the 1960's and 1970's, and these have come to be known as Evolutionary Strategies (ES's). Unlike GA's, the development of ES's was much more closely tied to traditional continuous stochastic optimization methods. Original ES's were developed as single-point search algorithms, that is a single population member. Variables are represented as real-valued numbers, to which mutation (ie, stochastic perturbation) is applied, commonly by adding values randomly drawn from a $\mathcal{N}(0,1)$ distribution. The extension to populations of samples and analogy with biology led to the incorporation of recombination and selection methods [11, 209, 210]. Again, the generic ES can be expressed as in Table 5.1, with these differences accepted.

The extension of the ES's to populations of size greater than one was explored by Schwefel [209]. In these cases, consider μ parents, at each generation producing λ offspring, with μ individuals being selected as parents for the next generation. The distinction is made between the $(\mu + \lambda)$-ES and the (μ, λ)-ES. In the former case, the parents are ranked and compete for survival into the next generation (and may survive indefinitely), whilst in the latter case the parents are completely replaced at each generation.

In the $(1, \lambda)$-ES, λ individuals are generated as mutations of a single parent solution. The offspring with the lowest error function value is retained as the parent for the next generation.

The $(1 + \lambda)$-ES is similar, the difference being that at each generation the parent is part of the competition for survival. There is no limit to the number of generations a member may

survive.

The field of Evolutionary Programming (EP) was developed independently by L. Fogel in the 1960's (see [76]). EP was originally developed for the evolution of finite state machines applied to solving prediction problems. The application of EP's to real-valued optimization tasks has revealed that the procedure is essentially equivalent to the ES's, with some variations on the implementation of evolutionary operators. EP is not considered further in this thesis.

5.1.3 Population-Based Incremental Learning

S. Baluja's Population-Based Incremental Learning (PBIL) algorithm was proposed as an optimization method which "removes the genetics from the genetic algorithm" [16, 19]. Rather than *maintaining a population of solutions* in the search space, PBIL *maintains a probabilistic model* of the promising regions of the search space, and uses this model to generate populations at each time step. Originally formulated for optimizing bitstrings, PBIL implements evolution through the construction of a probability vector

$$\mathbf{z} = (z_1, \ldots, z_n),\ z_i = P(x_i = 1)$$

where n is the length of the bitstring to be optimized, x_i is the i^{th} bit of a candidate solution bitstring, and z_i is the probability of the i^{th} bit in a bitstring being equal to 1. At each iteration of the algorithm, a population of bitstrings is generated by sampling from this distribution. Then, PBIL updates the probability model in the direction of the best member of the current population, governed by a learning rate parameter[1] η. This rule implements a "moving average" in the update of $\mathbf{z}$. For example, if $\eta = 0.01$ then at any time in the execution of the algorithm the current $\mathbf{z}$ depends on the previous $(0.01)^{-1} = 100$ generations. The dependency decays geometrically. Pseudo-code for a simple version of PBIL is given in Table 5.2. More sophisticated versions update towards the mean of the k best and away from the k worst members. A further variation is to mutate the probability vector after each update step. Experimental evidence has shown that PBIL can significantly outperform traditional genetic algorithms on a variety of test problems [17].

[1] This form of update rule is common: Baluja claims to have taken the inspiration for its use from the Hebbian rule of competitive learning [96].

Step 1:	Initialize probability vector $\mathbf{z} = (0.5, \ldots, 0.5)$
Step 2:	Generate pop. using prob. vector $D^{(k)} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$
Step 3:	Eval. cost fn. at each sample point $F^{(k)} = \{E(\mathbf{x}_1), \ldots, E(\mathbf{x}_k)\}$
Step 4:	Find best sample point in population $\mathbf{x}^* : f(\mathbf{x}^*) = min(D)$
Step 5:	Update probability vector $z_i = (1 - \eta)\mathbf{z}_i + \eta\mathbf{x}^*$
Step 6:	Goto Step 2 (until termination condition)

Table 5.2: The basic PBIL Algorithm.

Extensions to PBIL and Related Work

In [60], the Mutual-Information-Maximizing Input Clustering (MIMIC) algorithm is presented which improves on PBIL by modelling pairwise conditional probabilities between bits. Because PBIL models the probability distribution of each bit independently, MIMIC is able to offer improved performance on problems where the structure of the solution bitstring is of major importance, by capturing some of the relationships between bits in the bitstring. Baluja has also considered more sophisticated representations of pairwise conditional probabilities, where a tree of pairwise conditional probabilities is constructed (called an *Optimal Dependency-Tree*). Allowing only pairwise dependencies, the maximum spanning tree minimizes the (Kullback-Leibler) divergence between the true and estimated distributions [20].

MLP training was used by Baluja as one of the test problems for PBIL [17]. The standard PBIL representation was used, representing each weight as a bitstring with a finite given precision. A different application of PBIL to the MLP training problem is described in [77]. The algorithm was used to evolve network architectures, with a quasi-Newton method used to train each network. Cross-validation techniques were also used, in an effort to provide a hybrid algorithm producing nets with good generalization.

Real-valued PBIL

Baluja's original PBIL paper [16] makes a connection with the work in EA's for parameters with non-binary encodings, and suggests the extension of PBIL to real-valued parameter spaces,

using the methods developed in ES's.

This extension of PBIL to the optimization of functions of real-valued variables has only recently been considered. Servais et al. [213] consider using multiple-base encodings of solution variables. The same discrete probabilities are used, such that solution vectors encoded in base n will have a probability model for each variable consisting of $(n-1)$ real valued numbers. For example, a probability model for a single variable might be represented using a hexadecimal encoding of 16 numbers, representing $P(x=0), P(x=1), \ldots, P(x=\rho)$. The probabilities must sum to 1, that is

$$\sum_{i=0}^{16} P(x=i)$$

This encoding requires n numbers, although the complementation rule of probability allows this requirement to be reduced to $(n-1)$ numbers.

Servet et al. [214] propose and implement a Real-valued version of PBIL, using dichotomic uniform distributions. Assuming this probability model, each variable is constrained to some interval $[i_{Low}, i_{Up}]$, and each component z_i of the probability vector represents the probability that the i^{th} variable is greater than the midpoint of the interval[2], $(i_{Low} + i_{Up})/2$. Samples are drawn uniformly from each half-interval according to each z_i value. The probability vector is updated using the PBIL update rule, and the value of each z_i is checked. If z_i becomes close to 0 or 1 (for example, if $z_i > 0.9$ or $z_i < 0.1$), then the interval for variable i is halved in the appropriate direction ($i_{Low} = 0.5(i_{Low} + i_{Up})$ or $i_{Up} = 0.5(i_{Low} + i_{Up})$ respectively), z_i is reset to 0.5 and the algorithm continues.

Rudlof et al. [201] describe the first implementation of a real-valued version of PBIL. Independent univariate Gaussian distributions are used to model each variable of the solution vectors. Each Gaussian is specified by its mean, μ_i and variance, σ_i parameters. This model is a simplification over using the full Gaussian representing the joint probability distribution of the parameter vector. A bounded feasible region is defined as the search space, and the probability model is initially centered in this region. The mean value of each Gaussian is updated according to the PBIL update rule. Instead of simply using the single best sample to update the probability model, the implementation selects the k best samples and uses the average mean vector of these

[2]And the probability of the variable being in the lower half of the interval is given by $(1 - z_i)$.

individuals to update the model. The variance values are initialized to some large value, in an attempt to ensure that the feasible region is well-covered (in probability), and are then annealed towards some small value on a geometrically decaying schedule after every iteration

$$\begin{aligned} \sigma_i &= \sigma_i \gamma \, ; \\ \gamma &\leq 1. \end{aligned} \tag{5.1}$$

Sebag et al. [212] again consider using a vector of univariate Gaussian distributions as a probability model for Real-valued PBIL. The μ_i are updated by the PBIL rule and several different methods for updating the σ_i are discussed. The simplest choice is to fix all σ_i at some constant value, equivalent to setting the γ in 5.1 to unity.

Secondly, the σ_i can be adjusted in a manner similar to the method used to adjust variance values in ES's [209], that is

$$\sigma_i' = \sigma_i \cdot \exp(\tau' \cdot \mathcal{N}(0,1) + \tau \cdot \mathcal{N}_i(0,1))$$

where τ' and τ are constants. These variance values are perturbed in the same way as population points in an ES, with only indirect feedback received as to the success of variance permutation, through the success of the successively generated points.

Thirdly, the variance parameters can be adjusted on the basis of the sample variance of the K best members of the current population

$$\sigma_i = \sqrt{\frac{\sum_{j=1}^{K} \left(X_i^j - \overline{X}_i\right)^2}{K}}$$

where $\overline{X}$ is the sample mean of the best K offspring $X^1, \ldots, X^K$.

Finally, the previous method can be relaxed by an additional learning rate parameter α and updated similarly to the PBIL update rule:

$$\sigma_i^{t+1} = (1-\alpha)\sigma_i^t + \alpha \sqrt{\frac{\sum_{j=1}^{K} \left(X_i^j - \overline{X}_i\right)^2}{K}}$$

Sebag et al. compare the performance of each of these methods of variance adjustment on six artificial fitness functions of dimensionality 100 [212]. For these problems, the final method

provided the best performance in most cases.

5.1.4 Conceptual Ideas of Traditional Evolutionary and PBIL Algorithms

The field of EA's is largely concerned with algorithms which imitate some of the evolutionary processes found in nature. A population of individuals is normally created and evolved according to some evolutionary operators, and the picture of the dynamics of the algorithm is that of a group of individuals traversing the fitness landscape, searching for an optimum. As time goes on, new individuals are created and others die out, leading to the movement of the population towards regions of increasing fitness.

The intuition behind PBIL is quite different. As the algorithm is executed, the probability model is the only notion of the search that is maintained. Sampling from this distribution at each iteration creates a "population" of candidate solutions, but none of these solutions can "survive" into the next iteration of the algorithm. The sample is simply used to modify the model, in an attempt to make the sampling more successful on the next iteration.

Some further discussion of PBIL as an abstraction of a GA and a description of the dynamics of the algorithm can be found in [135].

5.2 Evolutionary Optimization and Probabilistic Modelling

In this section an algorithmic framework for population-based optimization algorithms is developed using some of the principles underlying PBIL. The basis of the framework is to *construct a probability distribution function that represents a model of the regions of the search space which are likely to lead to the global minimum (or high-quality solutions). The generation of search points by the algorithm then corresponds to sampling from this distribution. Furthermore, the resulting sample is used to update the probability distribution itself, refining the model in an iterative fashion*. The algorithm develops a probabilistic model of the search space. If successful, the probability model will "converge" on the global minimum, or at least will adjust itself such that sampling reveals high quality solutions with high probability. This behaviour is in contrast with the traditional view of optimization algorithms which are thought of as maintaining only the current point (e.g. random search, iterative improvement, simulated annealing (SA)) or a population of points (e.g. GA's, EP's, ES's). The probability model allows the focus

to be shifted slightly from simply attempting to find the global minimum, towards analyzing the dynamics of the search algorithm and the properties of the search space itself. The evolution of the probability model should be a powerful tool for this kind of analysis, which is otherwise difficult and has been little studied.

The conceptual distinction between PBIL and traditional EA's is the notion that what is maintained and evolved during the execution of the algorithm is a probabilistic model, rather than a finite population of solutions. The notion of a lasting population present in natural evolution, together with the traditional genetic operators (e.g. crossover) is removed in an attempt to provide faster, simpler and more robust optimization algorithms. What remains are the principles of

1. Parallel search using a population of solutions at each iteration of the algorithm;

2. Coupling of the current search results such that the information gained through the population is used to guide future search.

Cooperative search processes have been shown theoretically and experimentally to hold advantages over sequential/independent search [102, 109], indicating the potential for population-based, coupled optimization algorithms. Other research in this direction shares similar motivations [28], in particular cooperative and multi-agent parallel variants of simulated annealing (see, e.g. [36, 250] and the references therein). Cobb provides some discussion on the possibility of viewing GA's from this cooperative learning perspective [48].

In this chapter we explore the proposal that adopting this approach for EA's provides a useful perspective on their workings. From a more practical viewpoint, a general and common framework which follows this approach allows the development of new kinds of EA's.

5.2.1 Probabilistic Sampling Optimization

The optimization task (2.1) over a set of N Real-valued parameters collected into a vector $\mathbf{w}_i = (w_1, w_2, \ldots, w_N)$ is restated here for convenience

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

where $E(\mathbf{w})$ is the error or cost function. A general way of stating the search procedure for $\mathbf{w}^*$ is as follows.

Assume the existence of a probability density function $p(\mathbf{w})$, which obeys the usual conditions

$$p(\mathbf{w}) \geq 0$$

$$\int p(\mathbf{w}) d\mathbf{w} = 1$$

Initially, construct a probability density $p_0(\mathbf{w})$ that represents the model of the search space (or where it is believed that good solutions are distributed in $\mathbb{R}^n$). This initial density may be uniform, or it may be tailored to the specifics of a given problem, through the incorporation of prior knowledge. At each iteration (generation) t of the algorithm, a set D of k solutions is generated for the optimization problem by sampling k times from the probability density $p_t(\mathbf{w})$

$$D_t^{(k)} = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}.$$

For each sample solution (individual) the error function $E(\mathbf{w}_k)$ is evaluated, giving a corresponding set

$$F_t^{(k)} = \{E(\mathbf{w}_{1,t}), \ldots, E(\mathbf{w}_{k,t})\}.$$

The probability model is then modified according to some function Φ. This modification generates the probability model for the next generation, $p_{t+1}(\mathbf{w})$, and the process repeats. Pseudocode for this general framework is shown in Table 5.3.

Φ describes how the model of the search space is modified from time t to time $t+1$. Φ may depend on other known quantities, such as the probability model at previous generations, the samples (or individuals) of previous generations and their corresponding F values

$$\Phi \equiv \Phi(p_i(\mathbf{w}), D_i^{(k)}, F_i^{(k)}),\ 0 \leq i < t$$

In this way a model of the search space is modified as part of the optimization procedure. In the following it is shown that two very simple concepts from probability methods can be

Step 1:	`Initialize probability model`
	$p_0(\mathbf{x}),\ t = 0$
Step 2:	`Sample from` $p_t(\mathbf{x})$
	$D^{(k)} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$
Step 3:	`Evaluate`
	$F^{(k)} = \{E(\mathbf{x}_1), \ldots, E(\mathbf{x}_k)\}$
Step 4:	`Update probability model`
	$\Phi : p_t(\mathbf{x}) \rightarrow p_{t+1}(\mathbf{x})$
Step 5:	`Increment time`
	$t = t + 1$
Step 6:	`Goto Step 2 (until termination condition)`

Table 5.3: General Probabilistic Population-Based Algorithm.

represented in this framework.

Example: Uniform Random Search

As an example, consider random search over the entire space, where the probability density $p_t(\mathbf{w})$ is only sampled from once at each iteration ($k = 1$) and the sample $D_t^{(k)}$ becomes a single solution vector $D_t^{(1)} = \{\mathbf{w}_{1,t}\} \equiv \mathbf{w}_t$. The initial probability model is the uniform density over the entire search space

$$p_0(\mathbf{w}) = U(-\mu, +\mu),\ \mu \rightarrow \infty$$

and remains constant (i.e. Φ is an identity mapping/function of $p_t(\mathbf{w})$ only)

$$\Phi : p_{t+1}(\mathbf{w}) = p_t(\mathbf{w}).$$

Example: Random Walk

A simple random walk is quite similar to random search, except that a "neighborhood" is defined around the current point, and the next iteration produces a solution in this neighborhood (a "step" from the current point). If the neighborhood is defined through a Gaussian distribution

with zero mean and unit variance, and start the search at the origin: (again, $k = 1$)

$$p_0(\mathbf{w}) = \mathcal{N}(0, 1)$$

the probability model is updated so that at the next iteration, the Gaussian is centered on the current generated solution

$$\Phi : p_{t+1}(\mathbf{w}) = \mathcal{N}(\mathbf{w}_t, 1).$$

Random search and random walk methods do not consider the error function values for the solutions generated at each generation (ie, the "Evaluation" part of step 2 of the general algorithm is unnecessary). The above random walk can be modified to utilize unit variance Gaussian steps, so that the probability model (ie, move the center of the Gaussian) is only updated to the currently generated point if the value of the error function at the new point is less than the value of the error function at the current point. This modification leads to the following *iterative improvement* algorithm

$$p_0(\mathbf{w}) = \mathcal{N}(0, 1)(\text{as before}) \tag{5.2}$$

$$\Phi : p_{t+1}(\mathbf{w}) = \begin{cases} \mathcal{N}(\mathbf{w}_t, 1) & \text{if } F(\mathbf{w}_t) < F(\mathbf{w}_{t-1}) \\ p_t(\mathbf{w}) & \text{otherwise.} \end{cases} \tag{5.3}$$

Sampling from a Gaussian neighborhood means that most steps in the algorithm will be quite close to the current point, with occasional larger steps. Note that this algorithm is equivalent to *greedy search* with the given neighborhood definition. A criterion for deciding when to terminate such a search would also in practice be defined (e.g. stop if the last 1000 trials have resulted in no successful moves).

5.2.2 Evolutionary Strategies and Probabilistic Modelling

Although they are not considered in this manner, EA's for optimization are closely related to the above algorithmic framework. In this section it is shown how ES's can be expressed as population-based, probabilistic modelling algorithms.

Evolutionary Strategies - population size = 1

The (1+1)-ES is the special case where the algorithm produces a single offspring from a single parent, at each generation of the algorithm[3]. The offspring is the parent subject to some perturbation or mutation (often drawn from a unit variance Gaussian). Although the full covariance matrix is required to completely specify the (joint) N-dimensional Gaussian distribution, this requirement is usually simplified to variances only (diagonal covariance matrix), or even a single common variance for all components. In fact, the following statement can be made:

Result 5.1 *The simple (1+1)-ES is equivalent to the iterative improvement procedure described in Equations 5.2- 5.3 above, ignoring covariance terms.*

Multiple offspring: $\lambda > 1$

Maintaining a single Gaussian as the probability model (as for the (1+1)-ES formulation), only the $k(=\lambda) = 1$ situation was considered, where $p_t(\mathbf{w})$ is sampled once at each t. According to the above framework, at each generation t, a set $D_t^{(k)}$ of individuals and their corresponding error function values $E_t^{(k)}, k > 1$ may be generated.

Denote by $\mathbf{w}_j^*$ the individual with the lowest error function value at generation j, that is

$$\mathbf{w}_j^* : F(\mathbf{w}_j^*) = \min_i(E_j^{(k)}),\ 1 \leq i \leq k$$

In the $(1, \lambda)$-ES, λ individuals are generated as mutations of a single parent solution. The single offspring with the lowest error function value is retained as the parent for the next generation.

Result 5.2 *The $(1, \lambda)$-ES can be described in the above algorithmic framework as*

$$p_0(\mathbf{w}) = \mathcal{N}(0, 1)$$

$$\Phi : p_{t+1}(\mathbf{w}) = \mathcal{N}(\mathbf{w}_{*,t}, 1)$$

[3]A somewhat trivial algorithm might be called a (1,1)-ES, where the current point is mutated and the change is *always* accepted, resulting in precisely the random-walk behaviour discussed above. This procedure is not considered to be an evolutionary algorithm in any practical sense in the ES literature.

using an $\mathcal{N}(0,1)$ Gaussian as the initial probability model, which is centered on $\mathbf{w}_{*,t}$ for the t-th generation.

The $(1+\lambda)$-ES is similar, the difference being that at each generation the parent is part of the competition for survival.

Result 5.3 *The $(1+\lambda)$-ES can be described in the above algorithmic framework as*

$$p_0(\mathbf{w}) = \mathcal{N}(0,1)$$

$$\Phi : p_{t+1}(\mathbf{w}) = \begin{cases} \mathcal{N}(\mathbf{w}_{*,t}, 1) & \textit{if } F(\mathbf{w}_{*,t}) < F(\mathbf{w}_{*,t}) \\ p_t(\mathbf{w}) & \textit{otherwise} \end{cases}$$

where $F(\mathbf{w}_t^*)$ is in this case the mean of the Gaussian probability model of the previous generation. There is no limit to the number of generations a member may survive.

Multiple parents and offspring: $\mu, \lambda > 1$

The further extension is to the (μ,λ)-ES and $(\mu+\lambda)$-ES, which allow for multiple parents and offspring at each generation. At each generation, μ parents generate λ offspring via mutation (according to some probability density function). Offspring are ranked according to their error function values, and the best μ are retained as parents for the next generation (parents are included in the ranking for the $(\mu+\lambda)$-ES). To represent these possibilities in the above framework some well-known techniques of probability density estimation can be used.

Intuitively, the ES's with $\mu = \lambda > 1$ can be thought of as a kind of parallel search of the error surface, with each member of the population conducting its own iterative improvement search (each point being subject to Gaussian mutation/perturbation at each time step). The obvious way of modelling this search behaviour is to extend the above and centre a Gaussian function on each data point. Mutation then occurs by sampling from these Gaussians.

In the above framework, a number of independent probability sub-models can exist at each time step

$$p_t^1(\mathbf{w}), p_t^2(\mathbf{w}), \ldots, p_t^\mu(\mathbf{w})$$

and the update rule Φ for each $p_t^j(\mathbf{w})$ is analogous to the (1+1)-ES and the (1,1)-ES considered above.

Now, consider the (μ+1)-ES case[4]. A number of Gaussians, μ, can be maintained and one of them chosen uniformly to sample from (i.e. mutating one parent to produce an offspring). If the sample is not the worst (i.e. if the sample is better than any of the μ surviving parents), it becomes a parent for the next generation, and the worst offspring (mutated parent) is discarded. This effect can be achieved using a finite Gaussian Kernel or Parzen density estimator [32, 211], with μ kernels. This density is given by

$$p_t(\mathbf{w}) = \frac{1}{\mu} \sum_{i=1}^{\mu} \frac{1}{(2\pi h^2)^{d/2}} \exp\left\{ -\frac{\| \mathbf{w} - \mathbf{w}^i \|^2}{2h^2} \right\}$$

where h is an adjustable parameter, which controls the width of the Gaussian centered at each data point. This (joint) density is the summation of all the component Gaussians, each being equally weighted. In practice, a sample is drawn from the density at each generation by first choosing a kernel uniformly, and then sampling from that Gaussian. Starting with all μ kernels arbitrarily centered at the origin, the (μ+1)-ES can be expressed using this probability model.

Result 5.4 *The (μ+1)-ES can be described in the above algorithmic framework as*

$$p_0(\mathbf{w}) = \frac{1}{\mu} \sum_{i=1}^{\mu} \frac{1}{(2\pi h^2)^{d/2}} \exp\left\{ -\frac{\mathbf{w}^2}{2h^2} \right\}$$

$$\Phi : p_{t+1}(\mathbf{w}) = \frac{1}{\mu} \sum_{i=1}^{\mu} \frac{1}{(2\pi h^2)^{d/2}} \exp\left\{ -\frac{\| \mathbf{w} - \mathbf{w}_{*,t}^i \|^2}{2h^2} \right\}$$

where $k = 1$.

A Gaussian kernel density estimation model can be similarly applied for any number of parents and offspring. If $\mu > \lambda$, the probability model is sampled λ times at each generation, meaning that a random subset of the kernels are sampled, leading to the ($\mu + \lambda$)-ES[5]. When $\mu < \lambda$, some

[4]The (μ,1) case is impossible without additional operators.

[5]Again, (μ, λ)-ES for $\mu > \lambda$ is not possible without some additional operations, such as a crossover or recombination method.

kernels will be sampled more than once at each generation, but only μ kernels are formed for the next generation, whether the parents are permitted to survive (($\mu + \lambda$)-ES), or not ((μ, λ)-ES).

Result 5.5 *The ($\mu + \lambda$)-ES and (μ, λ)-ES (for $\mu \leq \lambda$) can be described in the above algorithmic framework.*

5.2.3 The Relationship Between PBIL and ES using Probabilistic Modelling

Consider the PBIL algorithm with a sample size of 1. This algorithm is similar to the random walk procedure above - the "parent" of one generation does not usually survive into the next, as there is only an infinitesimal probability of the sampling in the current PBIL generation producing a sample which was generated at the previous generation. The difference, however, is that PBIL explicitly represents and adapts $p_t(\mathbf{w})$ during the search process, whereas in (1,1)-ES, $p_t(\mathbf{w})$ is expressed as a mutation operator, and the intuition regarding the operation of the algorithm is that at any given iteration of the algorithm, only the single current member of the population is retained. PBIL can be reduced to the random walk procedure by choosing a population size of 1 and setting the learning rate parameter to 1.0. This reduction means that PBIL updates its probability model to completely reflect the (single) previous population member

$$\begin{aligned} p_0(\mathbf{w}) &= \mathcal{N}(0,1)(centered\ arbitrarily\ at\ the\ origin) \\ \Phi : p_{t+1}(\mathbf{w}) &= \mathcal{N}(\mathbf{w}_t, 1) \end{aligned}$$

Real-valued PBIL using Gaussian distributions can then be stated as

Result 5.6 *Real-valued PBIL is given by*

$$\begin{aligned} p_0(\mathbf{x}) &= \mathcal{N}(\mu_0, \sigma) \\ \Phi : p_{t+1}(\mathbf{x}) &= \mathcal{N}(\mu_{i,t+1}, \sigma); \\ \textit{with } \mu_{i,t+1} &= (1-\eta)\mu_{i,t} + \eta \mathbf{x}_t^* \end{aligned}$$

5.3 Optimization and Flexible Probability Modelling

In this section, the simple probability model underlying many existing EA's is highlighted and the algorithmic framework developed in this chapter is used to apply a mixture model density estimation technique to create an evolutionary optimization algorithm. This application is used to explore some of the features of this algorithm and to compare these features with other more familiar algorithms.

5.3.1 The Modelling Capabilities of EA's and PBIL

Although EA's have been successfully applied to many real-world optimization problems, interest in quantifying what kind of problems are easy or hard for EA's to solve has increased. This interest is partly due to the No Free Lunch (NFL) theorems, which indicate that there is no EA that outperforms all other algorithms in any general sense [255]. One limitation of some EA's (particularly GA's) is that each variable to be optimized is typically treated independently by the algorithm. For example, in a simple GA, each bit in an individual is mutated independently with some given probability. Furthermore, many of the functions which are used to test EA's can be linearly separated, and each variable optimized independently. EA's display a significant decrease in performance by simply rotating the objective function [205]. Also, on linearly decomposable functions, EA's display $O(N \ln N)$ complexity, whereas optimizing each variable independently could be done in $O(N)$ time [205, 206]. More advanced ES's evolve rotation angles for each variable, which allow them to overcome this difficulty [11].

A measure of the extent to which functions can be linearly decomposed, known as *epistasis* has been proposed as a measure of the difficulty of a function to be optimized by a GA in binary spaces. Unfortunately, the only relationship which seems to hold is that high epistasis/correlation between variables implies a function is GA-easy, limiting the applicability of the measure [195].

Previous implementations of PBIL for continuous search spaces have also used a very simple underlying probability model. Both the dichotomic model [214] and the Gaussian models [201, 212] follow the original binary-PBIL and model each variable independently using a very simple probability distribution. In the case of the Gaussian models, this implementation can be thought of as using univariate Gaussians for each variable, or a multivariate Gaussian

with a diagonal covariance matrix. The contours of such a Gaussian are constrained to be aligned with the coordinate/variable axes. Thus, the model is inadequate for capturing dependencies between variables in the search space, analogous to the binary-PBIL case. It is also not capable of modelling multimodal distributions or distributions with non-Normal characteristics.

The problem of extending Real-valued PBIL to incorporate more powerful and flexible probability models is somewhat different to the discrete problem. The techniques used in modelling pairwise discrete probabilities [20, 60] are not applicable to continuous distributions. The above framework for population-based probabilistic sampling algorithms however makes it clear that many of the techniques of probability density estimation, unsupervised learning and clustering might be successfully applied to this problem. It remains to discern what are the most important requirements of this problem domain, in order to compare and analyse the performance of these many unsupervised methods in this optimization problem setting.

There are several practical concerns with multivariate probability density estimation that are relevant to such a problem setting. Multivariate density estimation is itself a difficult problem, which suffers from the curse of dimensionality. It is usually impractical to attempt density estimation in a high-dimensional space, without having enormous quantities of data samples available (which would then be computationally challenging) [211]. Consider as an example using a single multivariate Gaussian distribution to model an N-dimensional density function. This distribution requires maintaining an N-dimensional vector of mean values, plus a full (symmetric) covariance matrix of $N(N+1)/2$ components [32]. Numerical sampling methods are available for a multivariate Gaussian distribution, but care must be taken to ensure that the covariance matrix does not become singular during execution of the algorithm. Achieving this becomes complicated for population-based optimization algorithms such as PBIL because it is desirable to be able to adapt the covariances as part of the stochastic search procedure. The situation is even more complex with a more flexible probability model (e.g. a mixture of full multivariate Gaussians).

One issue that is prevalent in the EA literature is the problem of maintaining diversity in a population during search. Over time, a population in an EA may be drawn to one region of the error surface by low cost function values. Depending on the properties of the landscape and the operators of the algorithm, this behaviour may occur at an undesirable time, such as very early in the execution of the algorithm. This effect may then result in premature convergence of

the population to a sub-optimal solution. Binary PBIL is also susceptible to this problem [18], where the model is not sufficiently flexible or well-matched enough to the situation to produce sufficiently diverse samples, and can become trapped in some region of the surface. Escape from this region will only occur after a very long time period.

One solution to this difficulty has been to remove the restriction of having a single, tightly coupled population, or to allow multiple populations or sub-populations. These algorithms include co-evolutionary [171] and parallel EA's [251]. Baluja [18] proposes a parallel version of PBIL (pPBIL), which displays a similar improvement over PBIL in performance for a number of test problems, as compared to using a parallel-GA over a standard GA.

5.3.2 Adaptive Gaussian Mixture Models

In this section the flexibility of the above algorithmic framework is demonstrated by implementing a powerful probability density estimation technique as part of a evolutionary optimization algorithm.

Gaussian mixture models represent a widely used probability density estimator. In this model the probability density $p(\mathbf{x})$ is represented by the sum of a number of component distributions

$$p(\mathbf{x}) = \sum_{j=1}^{M} \pi_j p(\mathbf{x}|j)$$

where M is the number of mixture components and π_j is the mixing coefficient for the j^{th} component. This model is similar to the kernel estimator, but the coefficients allow the contribution that each component makes to the sum to be varied, under the conditions

$$\sum_{j=1}^{M} \pi_j = 1$$
$$0 \leq \pi_j \leq 1.$$

Gaussian mixture models offer considerable flexibility in the forms of distribution they can adequately approximate. In particular, multimodal distributions can be effectively modeled (see, e.g. [32] for an introduction and further references). One approach to choosing the parameters

of a mixture model given a set of data is to use the Expectation-Maximization (EM) algorithm. This task represents an optimization problem in itself, hence factors such as local minima may cause difficulties in obtaining convergence [32]. Mixture models are the subject of ongoing research and a number of issues remain regarding their implementation.

The Adaptive Mixture Model algorithm of Priebe [185], which has some attractive properties is the particular technique used in this section. Firstly, it uses a recursive update rule (meaning that the model assumes that data points arrive sequentially, as they do in the algorithmic optimization framework of this chapter) to update the model after observing each new point (this step is the Φ procedure in the framework). These update equations are:

$$\begin{aligned}
\rho_{j,t+1} &= \pi_{j,t}\frac{p_t(\mathbf{x}_t^*|\pi_{j,t})}{p_t(\mathbf{x}_t^*)}, \\
\pi_{j,t+1} &= \pi_{j,t} + t^{-1}(\rho_{j,t+1} - \pi_{j,t}), \\
\mu_{j,t+1} &= \mu_{j,t+1} + \pi_{j,t}^{-1}t^{-1}\rho_{j,t+1}(x_{j,t}^* - \mu), \\
\sigma_{j,t+1}^2 &= \sigma_{j,t}^2 + \pi_{j,t}^{-1}t^{-1}\rho_{j,t+1} \\
&\quad \times((x_{j,t}^* - \mu_{j,t})(x_{j,t}^* - \mu_{j,t}) - \sigma_{j,t}^2).
\end{aligned}$$

where $\pi_{j,t+1}$ is the mixing coefficient, $\mu_{j,t+1}$ the mean and $\sigma_{j,t+1}^2$ the variance of the j^{th} component. $\rho_{j,t+1}$ represents the weighting of component j in the mixture.

Secondly, the number of mixture components, M, is allowed to vary during execution of the algorithm, beginning with a single component and adding further components if data is observed which is not adequately accounted for by the existing model. A simple approach suggested by Priebe [185] that is used here is to calculate the Mahalanobis distance from the new data point, $\mathbf{x}_t^*$, adding a new component if this distance is greater than some prespecified threshold value, T_C. In this case, the model is updated using

$$\begin{aligned}
\mu_{M_t+1,t+1} &= x_{j,t}^*, \\
\sigma_{M_t+1,t+1} &= \sigma_{0,t}, \\
\pi_{i,t+1} &= \pi_{i,t}(1 - t^{-1}), \\
\pi_{M_t+1,t+1} &= t^{-1}, \\
M_{t+1} &= M_t + 1.
\end{aligned}$$

A new component is assigned some small mixing coefficient, and the remaining coefficients are adjusted accordingly. An upper bound on M can be used to restrict the model to a manageable amount of component distributions.

5.3.3 Experimental Results

Two problems are used to demonstrate the application of the adaptive mixture estimator in an evolutionary optimization algorithm. These are the standard parabolic bowl

$$f_1(\mathbf{x}) = x_1^2 + x_2^2$$

and the well-known six-hump camel-back/Branin function [237]

$$f_2(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4.$$

f_1 is useful to demonstrate the convergence behaviour of the algorithm. f_2 is a multimodal cost function with local and global minima and so is able to show how the mixture model constructs a probabilistic search model which is able to capture the complex structure of the underlying search space.

For all algorithms used, the size of the population was set to 5 ($\lambda = 5$ for the $(1, \lambda)$-ES, and the search was initialized randomly in the region $[-5, 5]$. Each experiment was run for 1000 iterations/generations.

For problem f_1, four different algorithms were applied which are realized in the above framework:

- The Adaptive Mixture Model Algorithm (*AMix*) - maximum number of components in the mixture model = 10, initial variance of components new to the mixture = 2.0, Mahalanobis threshold for adding new components = 0.5.

- Real-valued PBIL using Gaussians ($PBIL_{CV}$) - fixed mean value of Gaussians = 1.0, fixed variance value of Gaussians = 1.0, $\eta = 1.0$.

- Real-valued PBIL using Gaussians ($PBIL_{Anneal}$) - fixed mean value of Gaussians = 1.0, initial variance value = 1.0, $\gamma = 0.82$, $\eta = 1.0$

Algorithm	f(**x**)
$AMix$	3.5e-7(2.9e-7)
$PBIL_{Anneal}$	0.9(1.4)
$PBIL_{CV}$	4.5e-6(3.6e-6)
ES	0.07(0.19)

Table 5.4: Results for the f_1 problem: mean(std. dev.).

- $(1, \lambda) - ES$ (*ES*) - standard Gaussian perturbations for each variable (initial variance of perturbations = 1.0), lognormal perturbations of step sizes using standard constants to control the overall ($\tau = \frac{1}{\sqrt{2\sqrt{n}}}$) and individual ($\tau' = \frac{1}{\sqrt{2n}}$) step sizes. For more details see, for example [210].

The averages over 10 runs of each algorithm are shown in Table 5.4. A typical run for each algorithm is shown in Figure 5.1. Although *AMix* gives the best results here (in terms of lowest error value reached), this performance is mainly artificial - parameters have been chosen for each algorithm to illustrate convergence behaviour. In addition, f_1 and f_2 are quite well-suited to modelling by Gaussian Mixtures. It is seen that *AMix* approaches the minimum at a rate $\sim 1/t$. In contrast, the constant variance value means that $PBIL_{CV}$ initially converges slowly due to the steps it takes being too small, increasing as the variance value becomes better suited to the local properties of the cost function, and then slows again as the variance becomes too large to generate the step sizes required to continue fast convergence. Annealing the variance ($PBIL_{Anneal}$) can result in convergence $\gg 1/t$; however, if γ is chosen incorrectly, it can also result in the convergence being "frozen", as shown here. The ES can also provide very fast convergence, but produces poor results with high variability in this instance. It is also worth noting that the computational effort required for each generation of the algorithms scales with the complexity of the probability model, since the model is updated at the end of each generation. Population size and the limited number of iterations are also important considerations.

Problem f_2 can be used to demonstrate the typical dynamics of the *AMix* algorithm. In the example run shown, 4 components are created in the mixture. Figure 5.2 shows the evolution of the mean values of the components (means for x_1 are shown as solid lines and for x_2 as dashed lines). Different components are seen to approach different mean values asymptotically. This problem has a pair of conjugate global minima at (0.08983,-0.7126) and (-0.08983,0.7126), the latter of which is approached by one of the components. This experiment shows how the Gaussian components in each dimension evolve to form a flexible multimodal density.

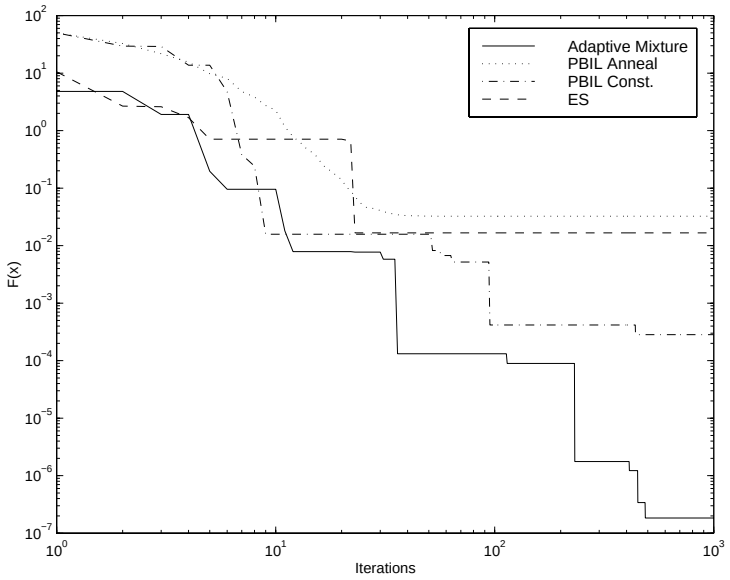

Figure 5.1: An illustrative run of each algorithm on f_1.

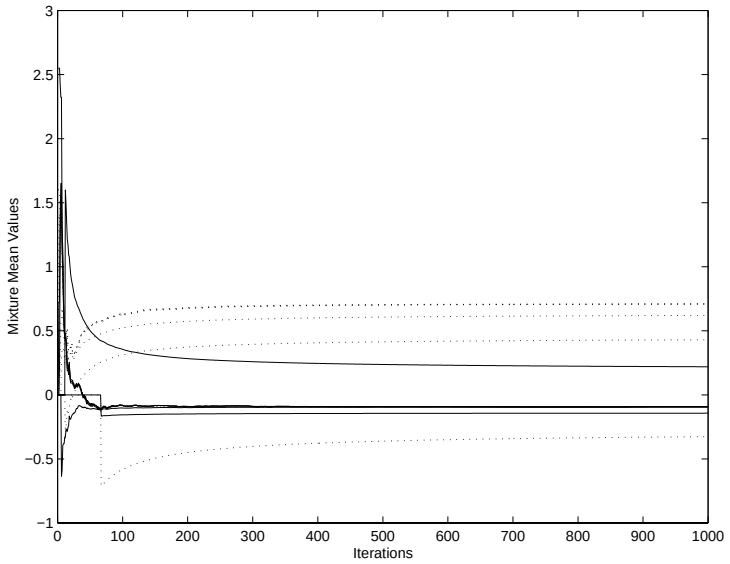

Figure 5.2: Evolution of mean values for a run of *AMix*. Each curve represents the mean value of the distribution for a single variable or dimension of the search space.

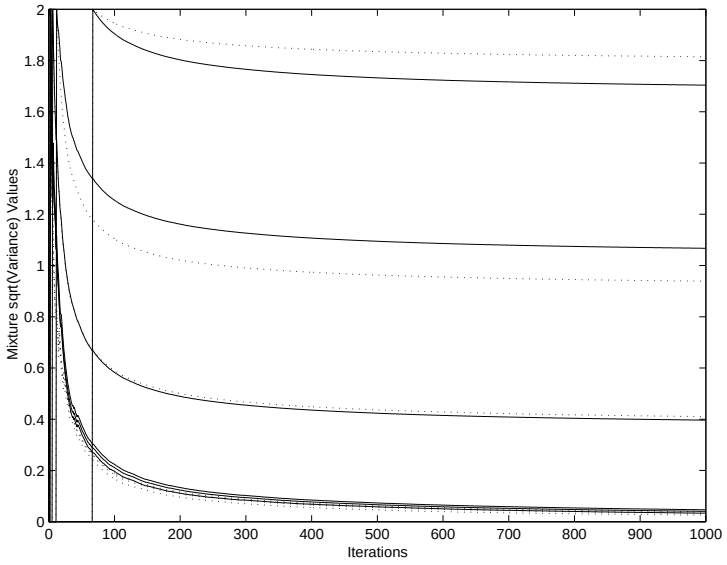

Figure 5.3: Evolution of the variance values for a run of *AMix*. Each curve represents the variance value of the distribution for a single variable or dimension of the search space.

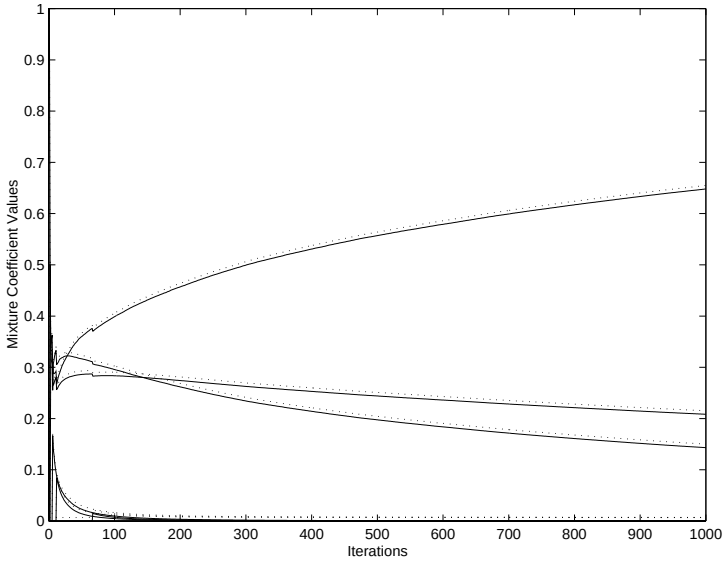

Figure 5.4: Evolution of the mixture coefficients for a run of *AMix*. Each curve represents the mixing coefficient strength value for a single Gaussian component of the distribution.

Figure 5.3 shows the evolution of the standard deviations of the components. Components can be observed leveling out to a range of different values (though the values for x_1 and x_2 are similar in each case). Some of the components can be seen to "converge" as their variances approach zero, while others remain at larger values. Finally in Figure 5.4 the mixing coefficients are shown. The component which approaches the global minimum is the topmost curve, becoming more heavily weighted in the mixture at the cost of the others. Two of the other components have significant mixing coefficients - these components correspond to the four highest variance curves in Figure 5.2. These components, whilst not converging towards either global minimum, remain part of the search by having corresponding variance values which allow them to produce a reasonable number of high-quality sample solutions. Other components are effectively removed from the model as their coefficient values become very small.

5.3.4 Discussion

This section has demonstrated the applicability of the algorithmic framework of this chapter through the application of a sophisticated method from probability density estimation - the Adaptive Mixture Model - to solving continuous optimization problems. It is clear that probability density estimation techniques may be "plugged-in" to this algorithmic framework without significant modification and without having to meet a large number of conditions or constraints. It is advantageous to use methods which are designed to work with on-line arrival of data, such as the Adaptive Mixture Model, though it seems that methods which use batch data could also be applied by changing the manner in which the data is collected and used to update the probability model.

The results of the experiments indicate that the optimization algorithm inherits some aspects of the density estimator. The t^{-1} factor in the Amix update equations allows the algorithm to focus the search and converge to the minimum of the parabolic bowl function at a rate proportional to this factor. On the Branin function, the algorithm is able to maintain search in multiple basins of attraction of minima, due to the structure of the mixture of Gaussians. However, these cost functions were chosen specifically to illustrate these features, and it is another matter to determine how the algorithm performs on a broader range of functions.

Further work is required to determine the practical utility of the AMix algorithm. The algorithm is somewhat more complex to implement than many existing EA's, and may be quite

sensitive to parameters such as the threshold for adding new mixture components. In the following section, a similar probabilistic model is used to develop another algorithm. This algorithm represents a logical extension to the underlying probability models used in ES and PBIL, whilst retaining much of the flexibility of the mixture model.

5.4 Modelling using Finite Gaussian Kernels

In this section a simpler extension to Real-PBIL is considered, using a finite Gaussian kernel probability density model. This extension also adds to the representational power of the model, and raises other issues which allow comparison to existing population-based algorithms. It can also be seen as an implementation of parallel, population-based, cooperative search.

In a finite Gaussian kernel density estimator, each variable is modeled by a sum of a number of Gaussian densities(see, e.g. [32]). While variables are still modelled independently, the finite kernel model adds significant power and flexibility to the forms of the density which can be effectively modelled. An important example is the ability to construct multimodal models of the error surface. This feature can be seen as analogous to allowing sub-populations in a EA, and consideration must be given to the coupling of the different kernels in the model.

A finite Gaussian kernel probability density estimator with m kernels can be written as

$$p(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^{m} \mathcal{N}(\mu_j, \sigma_j)$$

where m is the number of kernels and μ_m, σ_m are the mean and variance of the m^{th} kernels respectively.

Given this model, fixed σ_m values and the PBIL algorithm and update rule for the μ_m, an algorithm can be considered initially which is equivalent to the parallel execution of m *independent* PBIL algorithms with single Gaussian probability models. At each iteration, select uniformly between the m available Gaussians. The chosen model is then used to create samples and is updated by the PBIL rule, completing the iteration. The Gaussians will all move together as they are selected for iteration, and their sum forms the kernel probability model described.

Importantly, this algorithm is not significantly different, given t iterations of execution time,

to executing the m individual PBIL algorithms sequentially, giving each $\frac{t}{m}$ iterations[6]. This fact means that any exchange of information, or collective behaviour of the kernel model as a whole is not possible: each Gaussian behaves much like an independent "meta-individual".

In order to introduce the principle of cooperative search into the algorithm, consider a global shared storage mechanism. Each sample individual is created by first selecting a kernel at random, and then drawing from the kernel chosen. After the sample is collected, the cost function is evaluated for each sample in the population, and the best sample from each kernel is stored. Additionally, the single best sample from the entire population is recorded as the *global best* individual.

An additional global learning rate parameter is then introduced into the PBIL update rule. This rule becomes

$$\mu_{i,t+1} = ((1-\eta_g)\mu_{i,t} + \eta_g \mathbf{x}^*) + ((1-\eta_l)\mu_{i,t} + \eta_l \mathbf{x}_i^{lb})$$

where μ_i is the mean value of the i^{th} Gaussian, η_g is the global learning rate, η_l is the local learning rate, x^* is the global best individual, and x_i^{lb} is the local best individual which was drawn from the i^{th} kernel. The introduction of shared global storage, which is used to store x^* and which each kernel can use in its update rule, ensures that the kernels are no longer independently evolving.

Expressed in the notation of the above framework, this algorithm becomes:

Result 5.7

$$p_0(\mathbf{x}) = \frac{1}{m}\sum_{j=1}^{m}\mathcal{N}(\mu_j, \sigma_j)$$

$$\Phi : p_{t+1}(\mathbf{x}) = \frac{1}{m}\sum_{j=1}^{m}\mathcal{N}(\mu_{j,t}, \sigma_j)$$

$$\mu_{j,t+1} = ((1-\eta_g)\mu_{j,t} + \eta_g \mathbf{x}_{j,t}^*) + ((1-\eta_l)\mu_{j,t} + \eta_l \mathbf{x}_{j,t}^{lb})$$

[6]Random selection in the former case means that the time allocated to each kernel will only be approximately $\frac{t}{m}$.

which is applied to the general procedure outlined Table 5.3.

This algorithm is referred to as the Fink (**Fi**nite **n**umber of **k**ernels) algorithm. The addition of the second learning rate parameter, η_l allows the amount of local and global information to be controlled in the updates of the probability model mean values. For simplicity, variance values are set to a fixed constant value.

Implementation of the Fink algorithm is simpler than the AMix algorithm in several respects. Firstly, all kernels or components have equal weighting in the Fink model, eliminating the need to store and adapt mixture coefficients. Updating of the probability model in Fink is via the local and global PBIL-style equations above, which is significantly faster than the AMix update procedure. Finally, the capacity for adding components to the model is removed, further speeding up execution time.

5.4.1 Experimental Results: 2-D Test Problems

In this section the Fink algorithm is applied to some simple 2-D functions. These experiments allow the complete error surface to be visualized, and to demonstrate several features of the algorithm.

Parabolic Bowl

The first cost function considered is that of a 2-D quadratic bowl (This function is commonly referred to as the sphere model in the ES literature):

$$f_1(\mathbf{x}) = x_1^2 + x_2^2$$

This function is shown in Figure 5.5.

For this problem the search space was confined to the region $x_1, x_2 \in [-10, 10]$. The number of kernels m was set to 5. The mean values of each kernel were initialized randomly throughout the feasible search space, and the variance values were fixed at 1.0. The population size was set to 2, and the algorithm was run for 10^5 iterations or generations. Initially a small local learning rate was used ($\eta_l = 0.1$), and η_g was set to zero.

Figure 5.6 shows a curve of the best cost function value found so far, as a function of the number of iterations, for a typical training run. Also shown as points are the best single cost

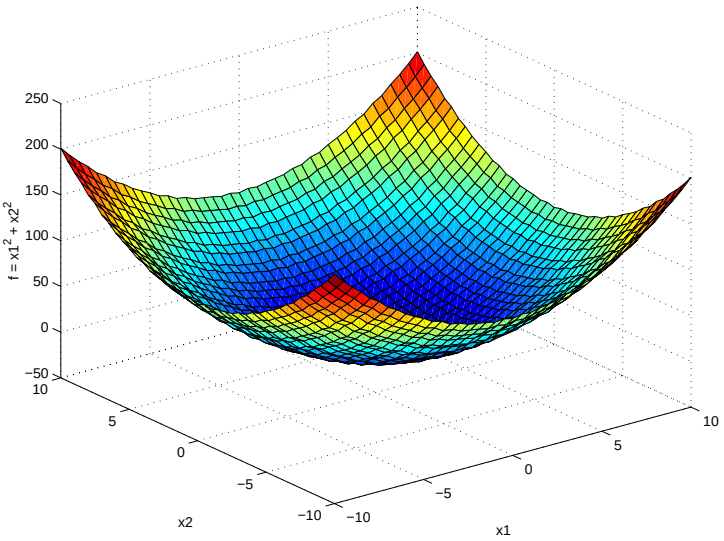

Figure 5.5: The 2-D parabolic bowl cost function.

function values found at each iteration. A decrease in the best-so-far curve indicates that the best cost value found on that iteration itself lies on the best-so-far curve. These points are not explicitly displayed in Figure 5.6 and in the figures below.

The initial probability model for this experiment is shown in Figure 5.7. Five Gaussian kernels can be seen scattered around the search space. After 10^5 iterations, the model has the form shown in Figure 5.8. It is clear that each kernel in the model has evolved towards the region of lowest cost at the centre of the bowl. The evolution of the mean value of each kernel is shown in Figure 5.9. Each kernel is observed to approach the minimum asymptotically from its different initial location. The smoothness of the cost surface is evident from these curves.

To explore the effect of varying the learning rate parameters in this simple example, 5 different configurations are considered, where $\eta_l = 0.1, 0.09, 0.05, 0.01, 0.0$ and $\eta_g = 0.0, 0.01, 0.05, 0.09, 0.1$, respectively. Each of these configurations was run 20 times from different random initializations. In addition, each 20-run batch was run with three different population sizes: 2, 5 and 50. The average performance of the Fink algorithm for each configuration is shown in Figures 5.10- 5.12. The upper solid line in each figure represents the result for $\eta_l = 0.1, \eta_g = 0.0$. The general trend across each figure is that convergence speed is increased as the global learning factor increases. As the population increases from Figure 5.10- 5.12, this difference in convergence speed diminishes. It is clear from knowledge of the cost function that the global information always leads to improved performance towards the global minimum. However from larger populations, the local best samples from each kernel are on average closer together.

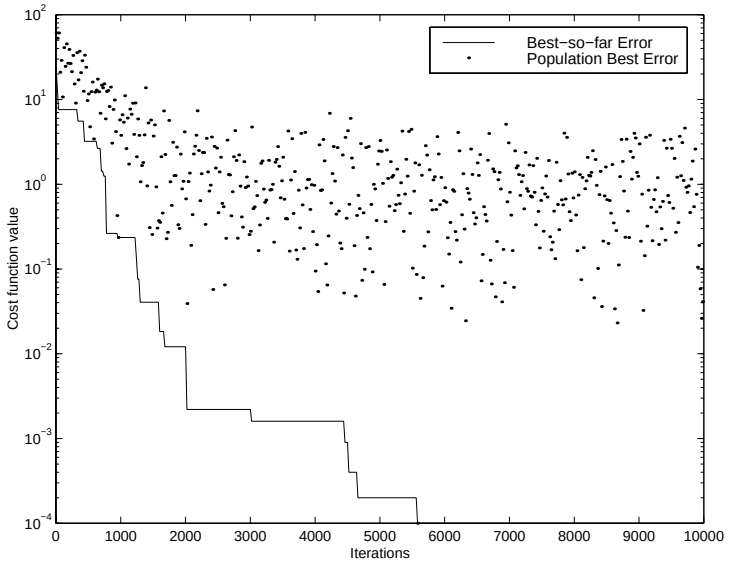

Figure 5.6: Typical performance curves for the parabolic bowl function.

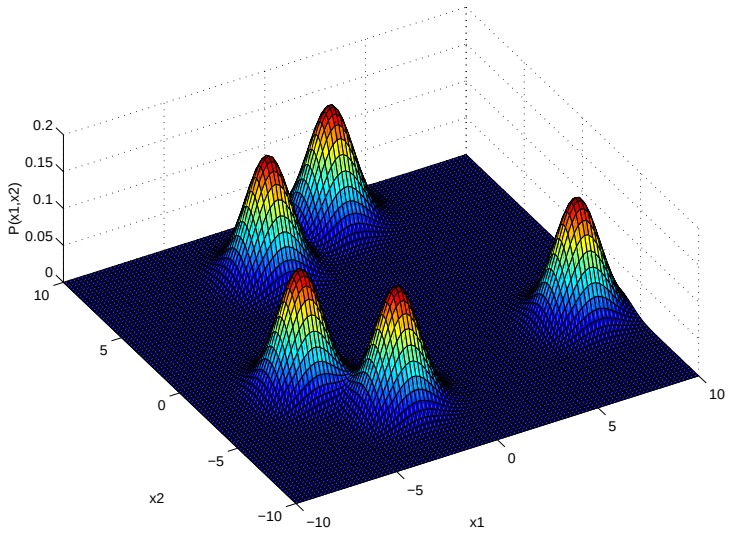

Figure 5.7: Initial probability model for the *Fink* algorithm on the parabolic bowl problem.

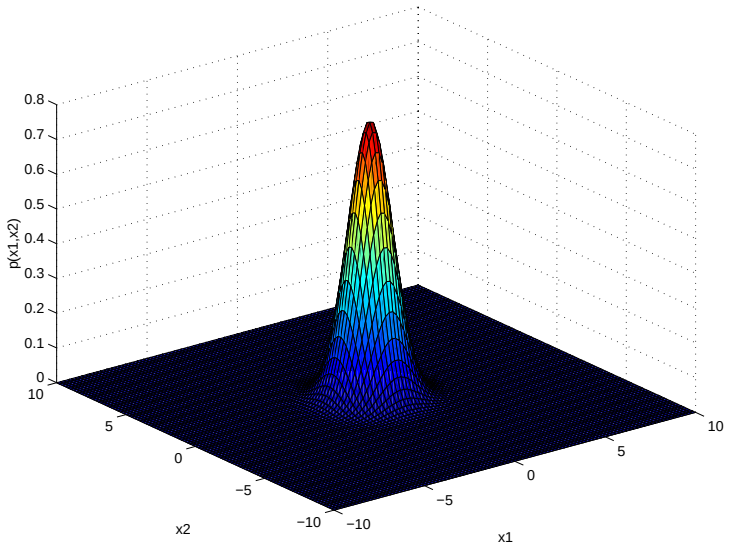

Figure 5.8: The probability model for the parabolic bowl problem after 10^5 iterations.

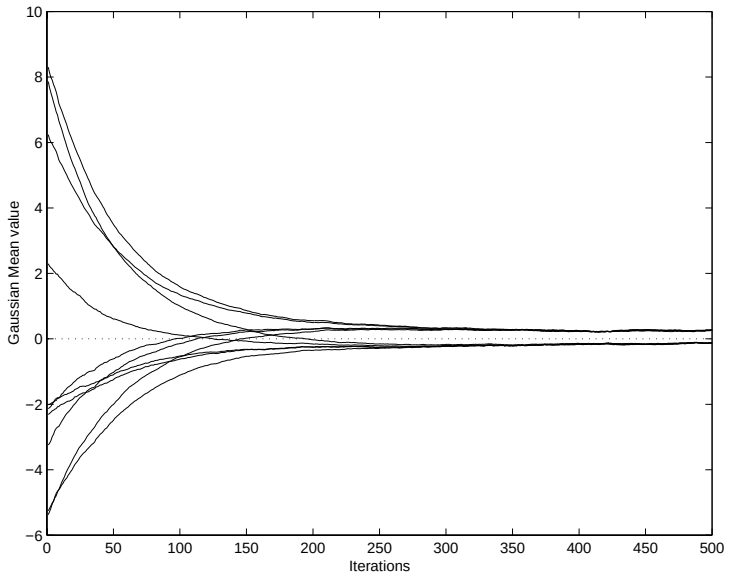

Figure 5.9: Evolution of the mean values for each kernel during the parabolic bowl example.

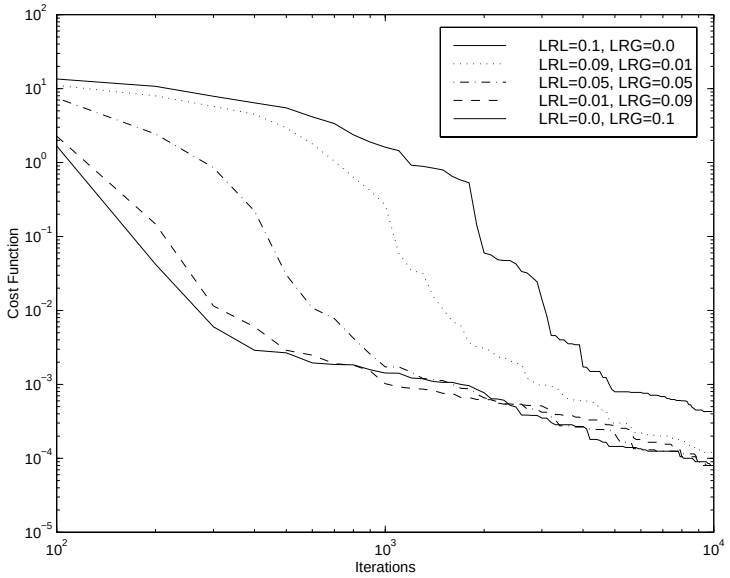

Figure 5.10: Average training curves for the Fink algorithm on the 2-D parabolic bowl, population size = 2.

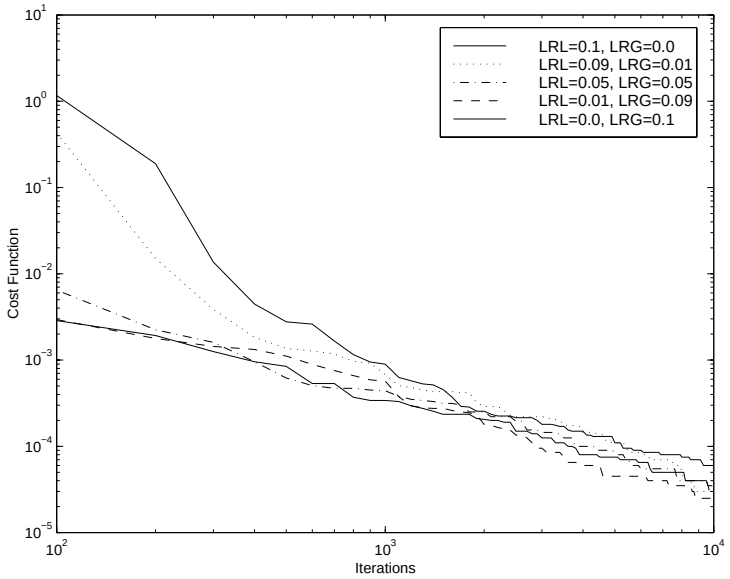

Figure 5.11: Average training curves for the Fink algorithm on the 2-D parabolic bowl, population size = 5.

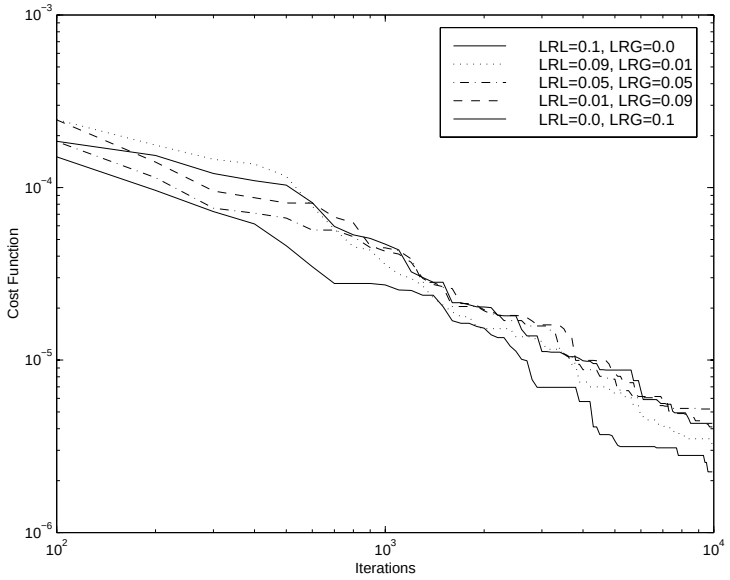

Figure 5.12: Average training curves for the Fink algorithm on the 2-D parabolic bowl, population size = 50.

Average curves are shown again with corresponding standard deviations for ($\eta_l = 0.0, \eta_g = 0.1$) and ($\eta_l = 0.1, \eta_g = 0.0$) in Figures 5.13- 5.15. These curves give an indication of the variability of the algorithm to randomness of initialization and during execution, for a local or global learning update. No significant difference is observable from these curves between the local and global learning update.

Rastrigin Function

The second 2-D problem tested is the Rastrigin function [237]:

$$f_2(\mathbf{x}) = x_1^2 + x_2^2 - \cos 18x_1 - \cos 18x_2.$$

This cost function (Figure 5.16) has about 50 local minima symmetrically arranged around the global minimum, in the feasible region chosen ($-1 \leq x_i \leq 1, i = 1, 2$). The global minimum is attained at the origin and its value is -2. The number of kernels in the model was set to 20. Other experimental settings were unchanged.

Typical performance curves showing best-so-far obtained error and best error values obtained at each iteration for this function are shown in Figure 5.17. As for the parabolic bowl, the initial and final probability models are shown in Figures 5.18 and 5.19 respectively. The larger number of kernels clearly shows how the algorithm constructs and adapts a probabilistic

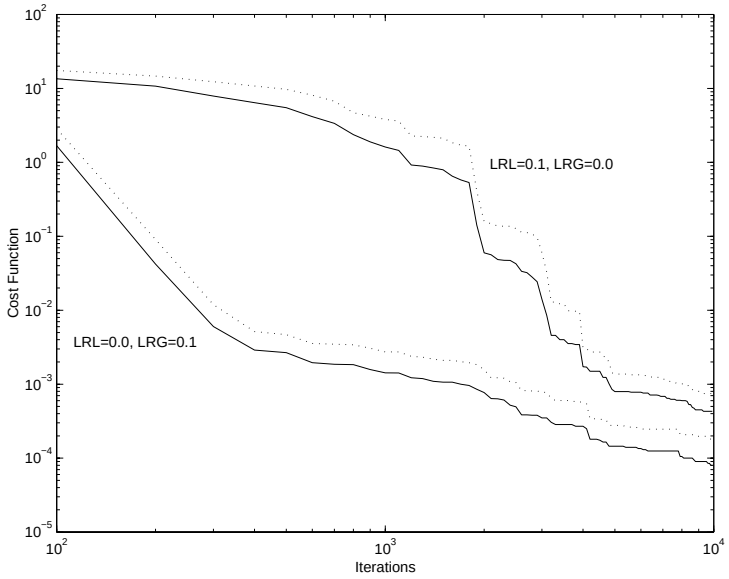

Figure 5.13: Average and Standard deviation curves for 2-D parabolic bowl Fink experiments, population size = 2.

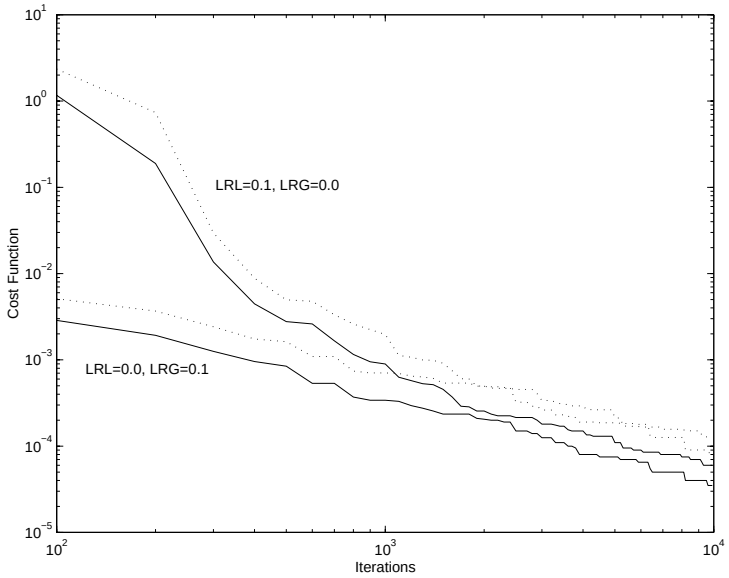

Figure 5.14: Average and Standard deviation curves for 2-D parabolic bowl Fink experiments, population size = 5.

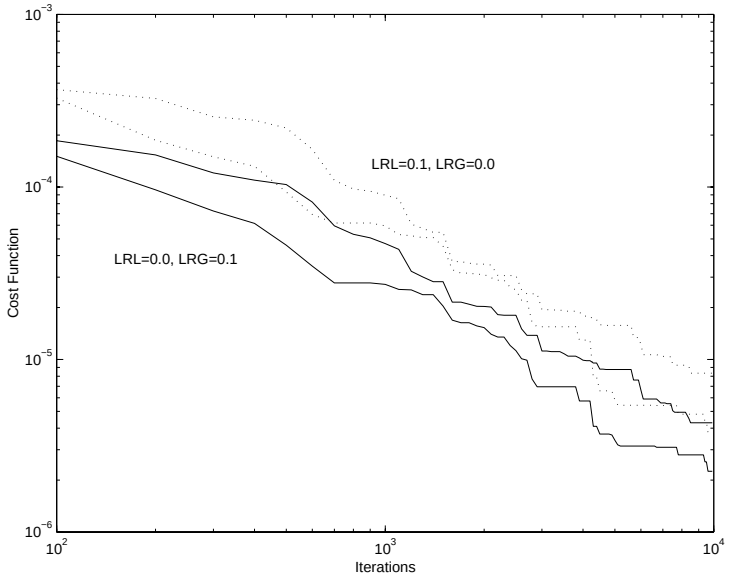

Figure 5.15: Average and Standard deviation curves for 2-D parabolic bowl Fink experiments, population size = 50.

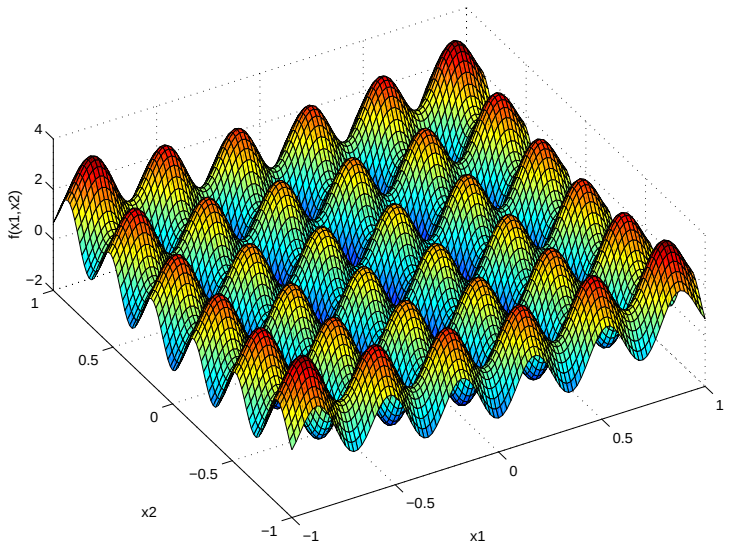

Figure 5.16: The 2-D Rastrigin cost function.

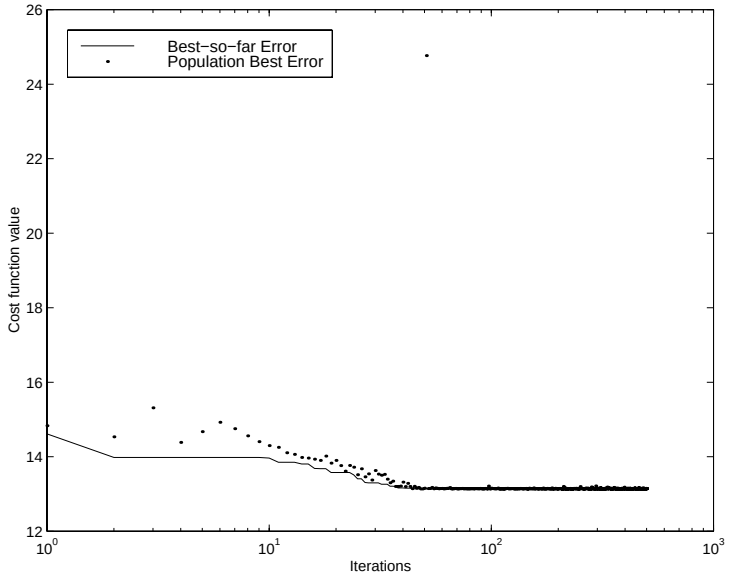

Figure 5.17: Typical performance curves for the Rastrigin function.

model of the cost function as part of the optimization process. The highly multimodal nature of this function is reflected in the final probability model. Importantly, the model is sufficiently capable of capturing this structure. The convergence of the mean values of the kernels, shown in Figure 5.20, displays the convergence of each kernel to the local minima of the function.

The average performance curves for this function (Figures 5.21 - 5.23) show how performance varies with different learning rates and population/sample sizes. The global learning factor can be thought of as a "greedy" factor which facilitates convergence of the kernels in the model to a single region of the search space. In contrast, the local learning factors can preserve diversity by allowing kernels to converge to local attractors of the search space. It is also important to recognize that the interaction between the cost function itself and the global/convergence and local/diversity elements plays a major part in determining the performance of the algorithm. Increasing the population also drives convergence of the algorithm, as this increase effectively increases the amount of "search" conducted by each kernel during each iteration of the algorithm. This effect leads to the discovery of lower cost values sooner, on average for each kernel. Figures 5.24 - 5.26 show average performance curves together with standard deviations for ($\eta_l = 0.0, \eta_g = 0.1$) and ($\eta_l = 0.1, \eta_g = 0.0$). As for the parabolic bowl experiments described above, no significant difference is observable from these curves between the local and global learning update. Further experiments are required to investigate any relationship between the local and global learning rates, the size of the population, the cost function and the variability

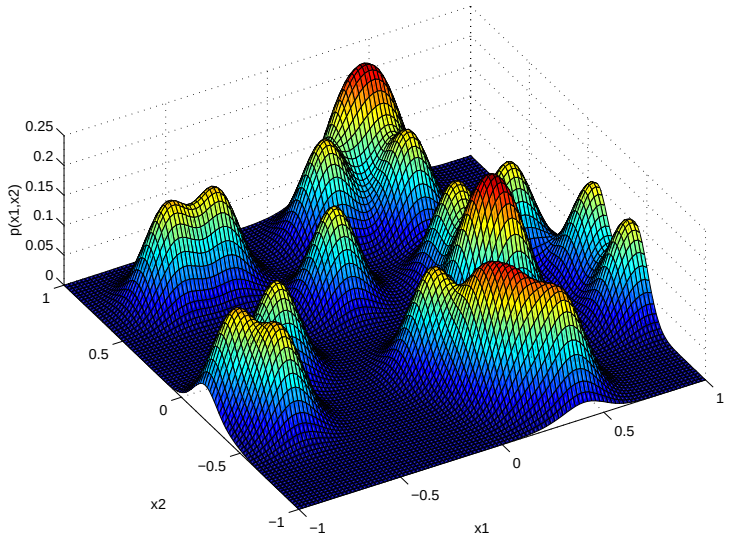

Figure 5.18: Initial probability model for the *Fink* algorithm on the Rastrigin problem.

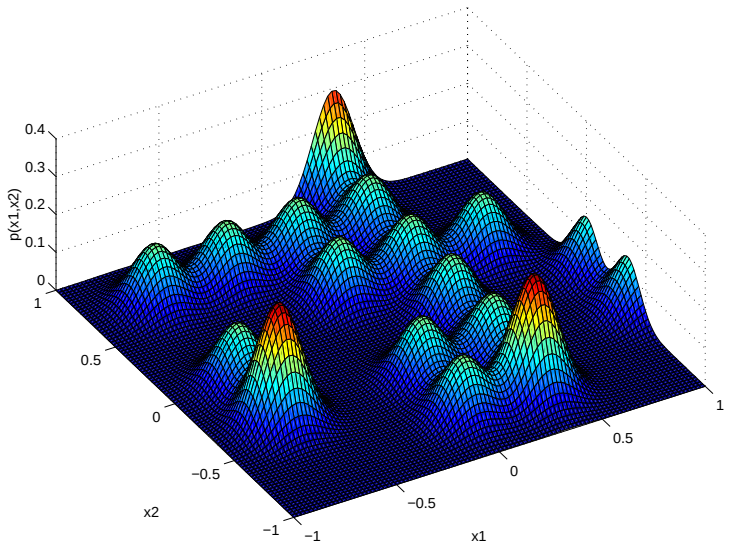

Figure 5.19: The probability model for the Rastrigin problem after 10^5 iterations.

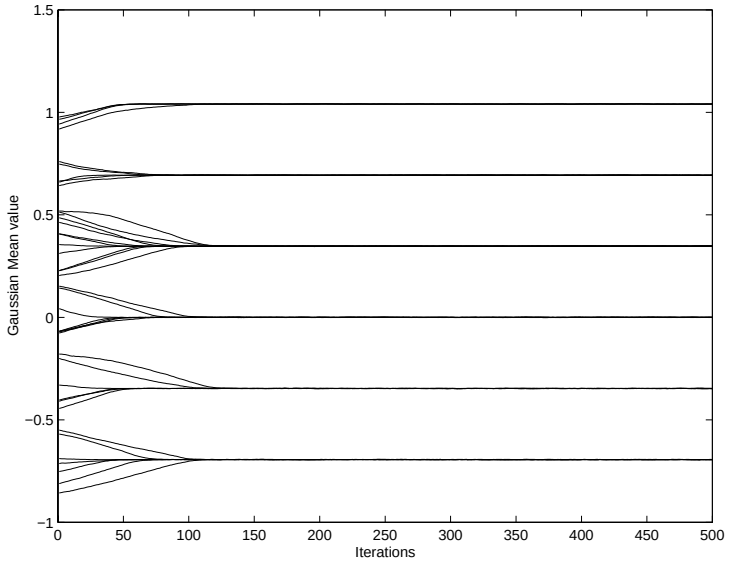

Figure 5.20: Evolution of the mean values for each kernel during the Rastrigin example.

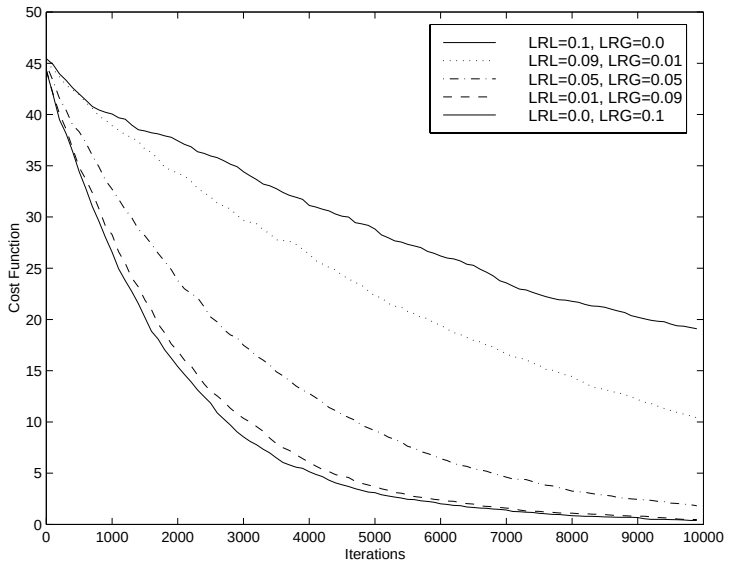

Figure 5.21: Average training curves for the Fink algorithm on the 2-D Rastrigin function, population size = 2.

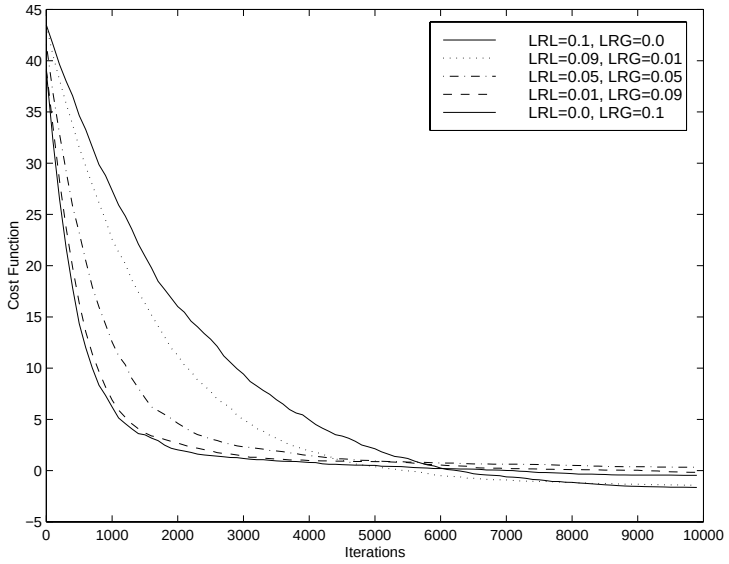

Figure 5.22: Average training curves for the Fink algorithm on the 2-D Rastrigin function, population size = 5.

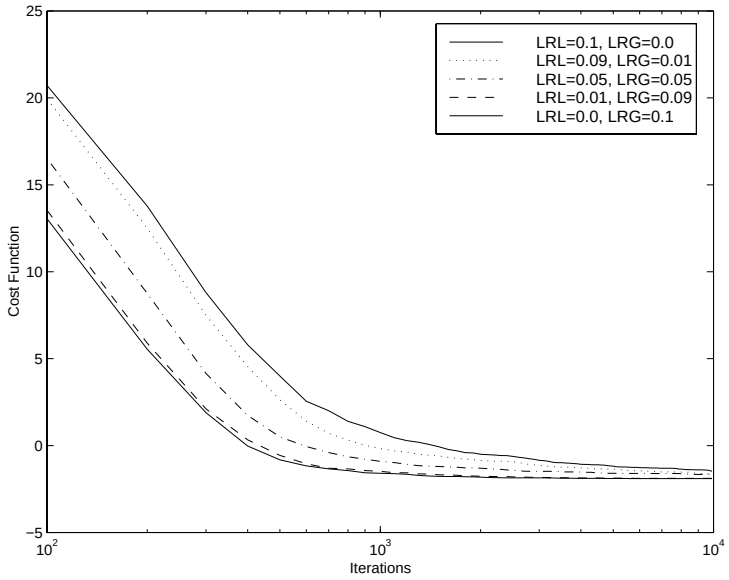

Figure 5.23: Average training curves for the Fink algorithm on the 2-D Rastrigin function, population size = 50.

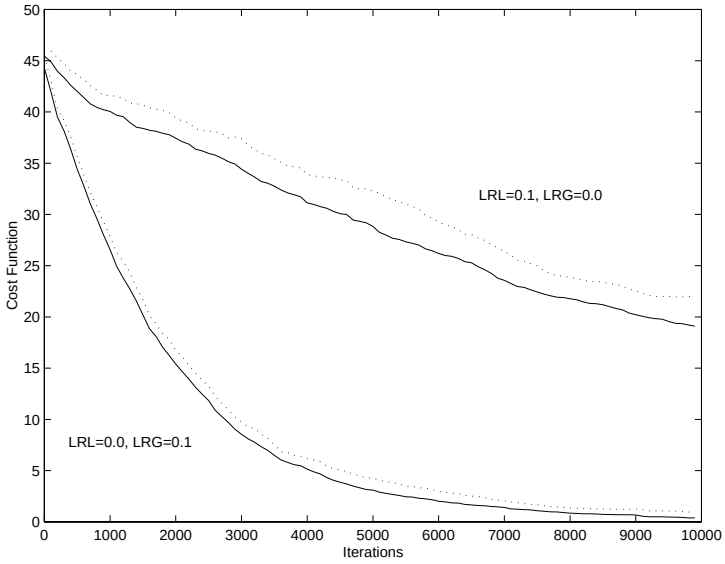

Figure 5.24: Average and Standard deviation curves for 2-D Rastrigin function Fink experiments, population size = 2.

of the algorithm to its random elements.

5.4.2 Training MLP's with the Fink Algorithm

In this section the Fink algorithm is applied to a number of different MLP training problems - the Exclusive-OR (XOR) problem (2-2-1 network), 4-bit and 8-bit encoders (4-2-4 and 8-2-8 networks), the cancer training problem (9-5-2 network) and the glass problem (9-9-6 network). Five different learning rate configurations were tested, as in the 2-D problems discussed above. Each experiment was run 20 times from different initial settings.

4-bit and 8-bit encoders

Previous work has shown that empirical results of this kind are often not well summarized by reporting the mean and standard deviation of the runs [137]. Box-whiskers plots [46, 47, 137, 239] are more robust to data whose distribution may be non-normal. Figure 5.27 shows the results for the 4-2-4 encoder experiments. Each Box-whisker glyph shows the Inter-Quartile Range (IQR) of the data as a box and the median as a line through this box. Whiskers extend to the maximum and minimum data values. Additionally, outliers (chosen to be points greater than 1.5 times the IQR) are shown separately.

In these experiments, the performance of the Fink algorithm tends to improve as the global

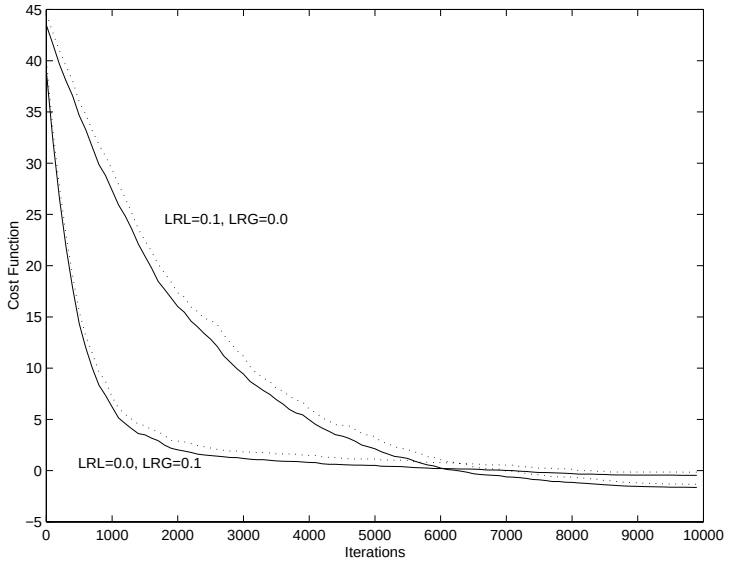

Figure 5.25: Average and Standard deviation curves for 2-D Rastrigin function Fink experiments, population size = 5.

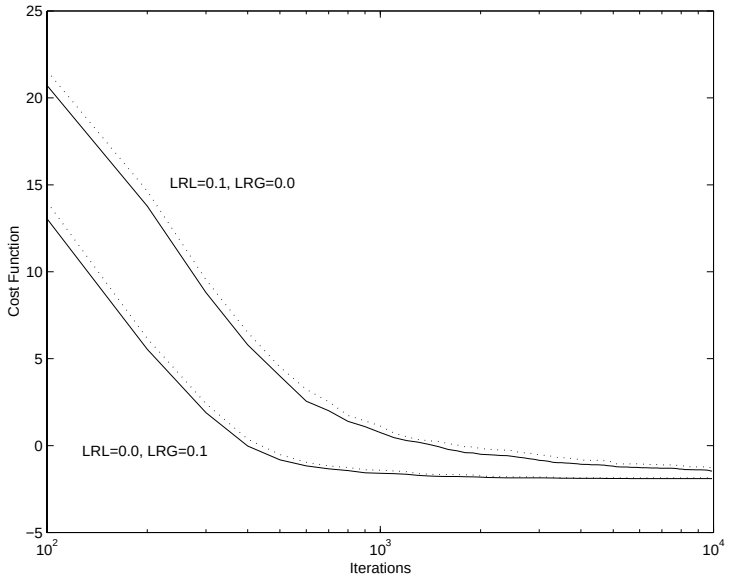

Figure 5.26: Average and Standard deviation curves for 2-D Rastrigin function Fink experiments, population size = 50.

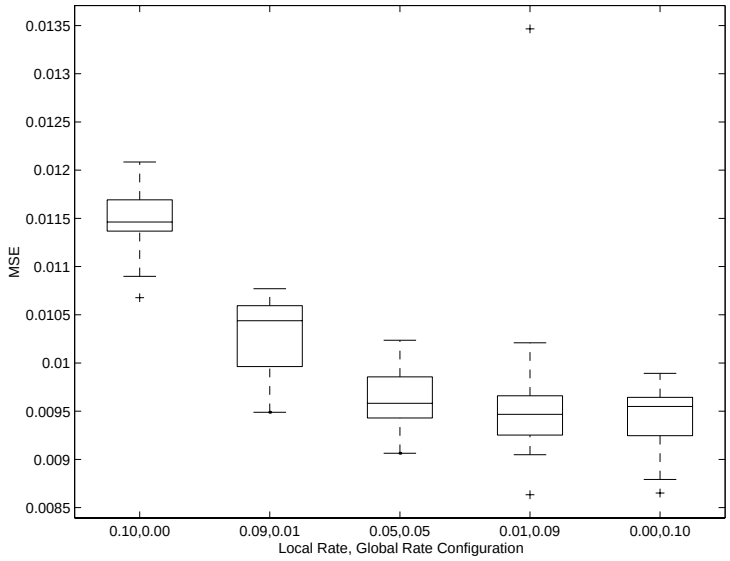

Figure 5.27: Summary of performance of Fink algorithm on the 4-2-4 MLP Encoder problem.

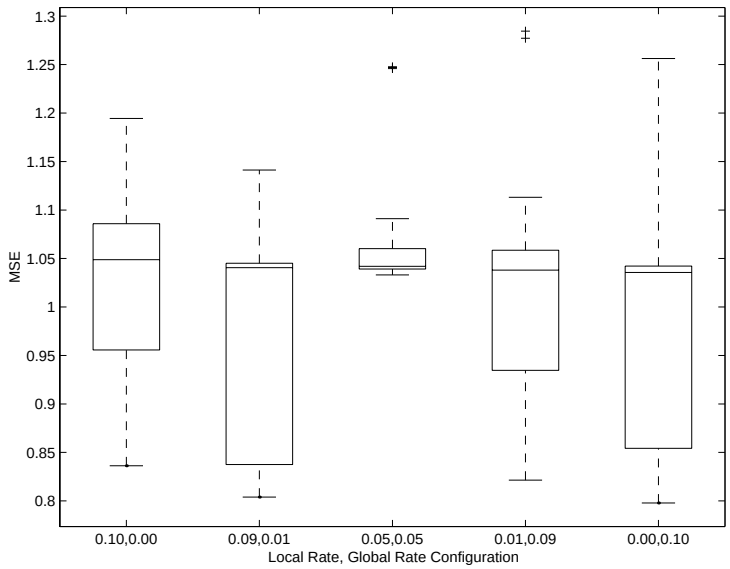

Figure 5.28: Summary of performance of Fink algorithm on the 8-2-8 MLP Encoder problem.

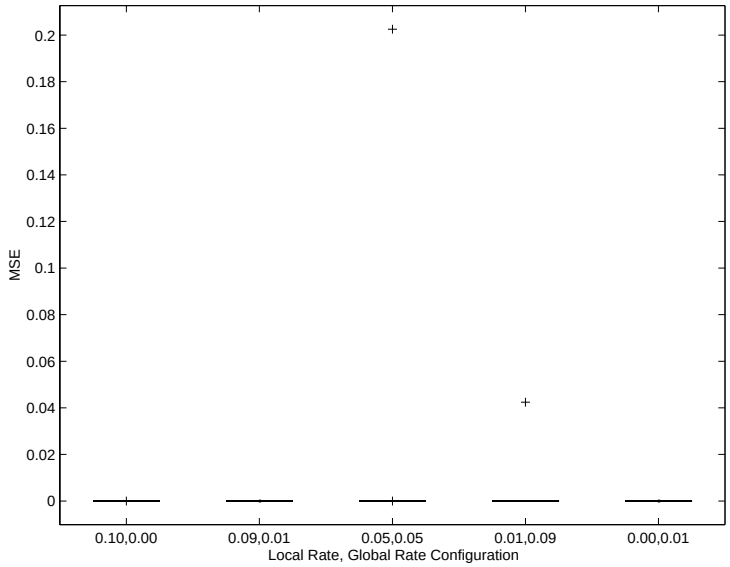

Figure 5.29: Summary of performance of Fink algorithm on the 2-2-1 MLP XOR problem.

learning rate is increased. Figure 5.27 also shows that the distribution of results for each learning rate configuration are reasonably similar, despite a few outliers. The impression is that the error surface is fairly well suited to a global descent approach, though convergence is not always guaranteed. The results for the 8-2-8 encoder (Figure 5.28) are in contrast to this. The box-whisker summary of the results clearly shows more variability, and the error rate attained is relatively poor in all cases. This result reflects the increasing difficulty of the 8-2-8 encoder compared to the 4-2-4 case. Adaptive variance or refinement of other algorithm parameters may lead to improved performance.

XOR

The Fink algorithm was also tested on the XOR training problem. Unfortunately, in this case the wide outliers shown make it impossible to interpret the distributions of the results at this scale (see Figure 5.29). A zoomed-in version of these results are shown in Figure 5.30. Most of these experiments converge to a very low error rate, with the distribution of results narrowing and improving slightly as the global learning factor is increased. The outliers indicate that the XOR error surface can be difficult to negotiate for this algorithm, which agrees with other results with different algorithms, for example backpropagation.

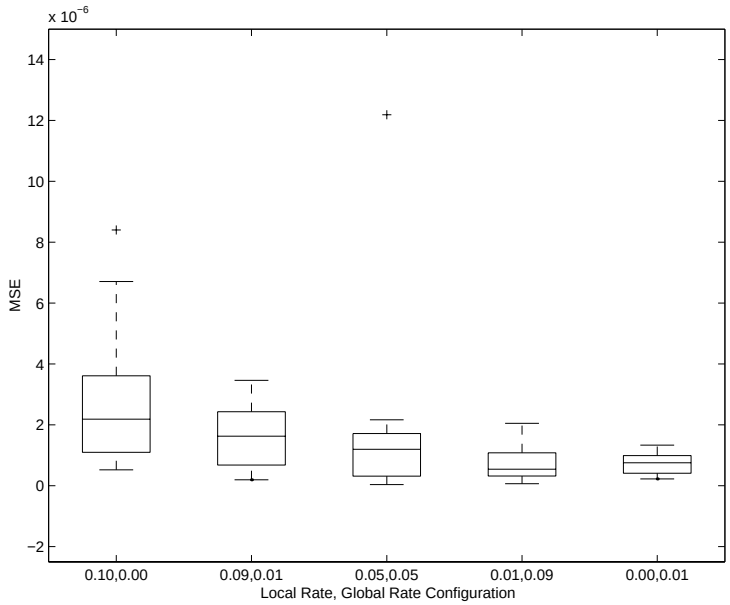

Figure 5.30: Summary of performance of Fink algorithm on the 2-2-1 MLP XOR problem after removal of outliers. Note the change in scale from Figure 5.30.

Cancer problem

The summary of results for the cancer problem are shown in Figure 5.31. This problem again produces result distributions which vary significantly with different learning rate settings. The level of MSE for these results is quite low. No trends are obvious with the variation of learning rates from completely local through to totally global.

Glass problem

The final test problem is the 9-9-6 MLP trained on the glass dataset. A summary of results is shown in Figure 5.32. For this problem the results suggest that a learning rate setting around ($\eta_l = 0.01, \eta_g = 0.09$) leads to the best performance. The distributions for each setting are approximately symmetrical and quite similar - the ($\eta_l = 0.1, \eta_g = 0.0$) case being a somewhat narrower distribution.

5.4.3 Visualization of the Fink Algorithm Probability Model

Algorithms which are based on the probabilistic modelling framework described in this chapter provide an obvious possible benefit for visualization purposes. This benefit is through concentrating visualization efforts on the evolution of the probabilistic model underlying the search,

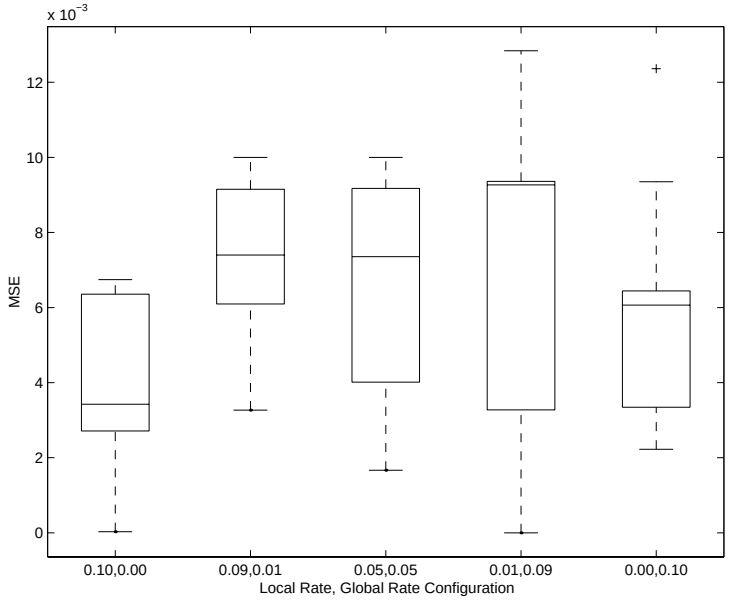

Figure 5.31: Summary of performance of Fink algorithm on the 9-5-2 MLP Cancer problem.

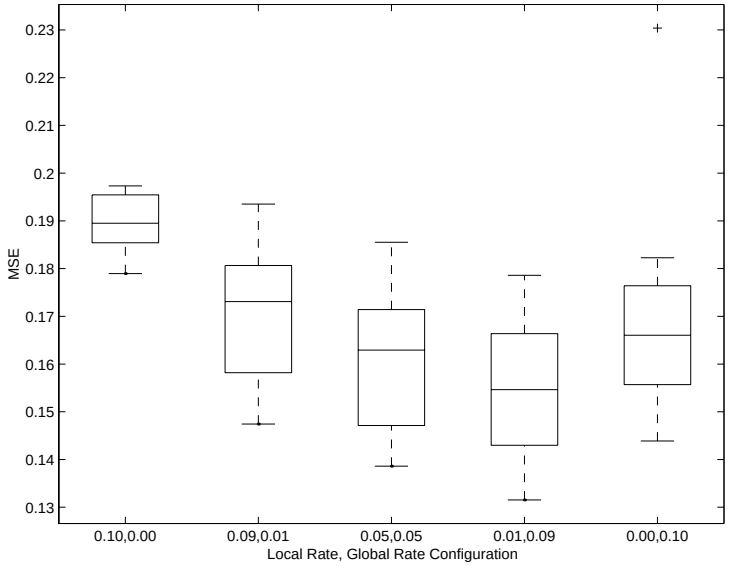

Figure 5.32: Summary of performance of Fink algorithm on the 9-9-6 MLP Glass problem.

rather than the stochastic trial points generated.

The evolution of the mean values of the Fink kernels can be plotted as in Section 5.4.1 above. When a moderate number of mean values are present it is useful to partition these values in some sensible way. For MLP training, this partitioning can be done by adopting from the Hinton diagram the idea of incorporating the structure of the network in the visualization.

Figure 5.33 shows the evolution of a single component of the Fink model for a 8-5-2 glass experiment. The visualization is organized according to the structure of the network, with one subgraph for each hidden and output unit in the network. The weights connected to the "fan-in" of inputs of each unit are shown in the corresponding subgraph. The upper row of the figure represents the hidden layer (5 units) and the lower layer represents the output layer (2 units). The structure of the network is incorporated into this visualization in a similar way to the Hinton diagram (Section 3.4.2). For this visualization, $N_i + 1$ curves will be drawn in each of the N_h subgraphs in the upper row, and $N_h + 1$ curves will be drawn in the N_o subgraphs in the lower layer. The evolution of the mean values for each weight are shown in Figure 5.33, with the mean of the bias weight distributions shown as dots. The stochastic nature of the algorithm is clear, and the evolution of different weights in the network can be compared. In this case,the second and third hidden layer subgraphs show quite similar behaviour in the evolution of the kernel means, suggesting that these weights are correlated to some extent. The figure also shows a tendency for weights connected to the inputs of the hidden units to develop to larger magnitudes that those in the second layer of weights connected to the output layer units.

Each curve in Figure 5.33 can be thought of as part of a learning trajectory for the probability model. Curves which increase or decrease reasonably smoothly indicate a similar feature of the error surface in that direction. Other curves flatten out, indicating little progress for the algorithm in the given direction, perhaps due to a flat part of the error surface. The visualization gives an indication of the features of the error surface which impact on the search process.

Figure 5.34 shows the evolution of *each* of the five components of the kernel model for an XOR experiment. The five subgraphs representing the kernels are not arranged in any particular order, but within each subplot the network structure is again reflected in the same way as Figure 5.33 above. This visualization allows comparison between the overall dynamics of the different kernels in the model. For example, plot 1 in the first kernel is somewhat similar to plot 1 in the fourth kernel, whereas plot 2 in the first and fourth kernels are quite different.

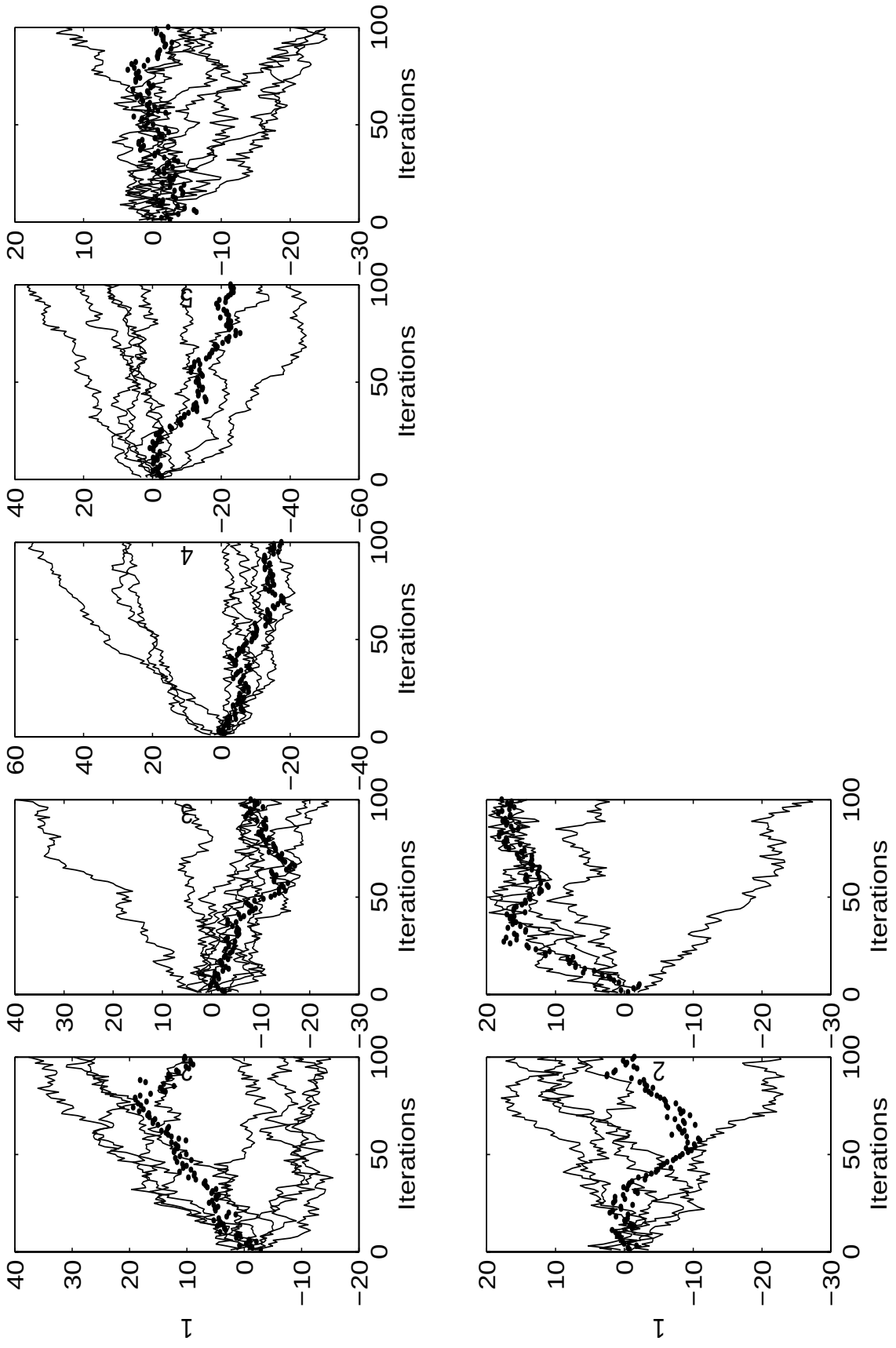

Figure 5.33: Evolution of Mean values of one kernel in a 8-5-2 glass experiment. Each curve represents the mean value of the distribution for a single variable or dimension of the search space. The vertical axes represents the mean value, with each subgraph labeled by the network unit number.

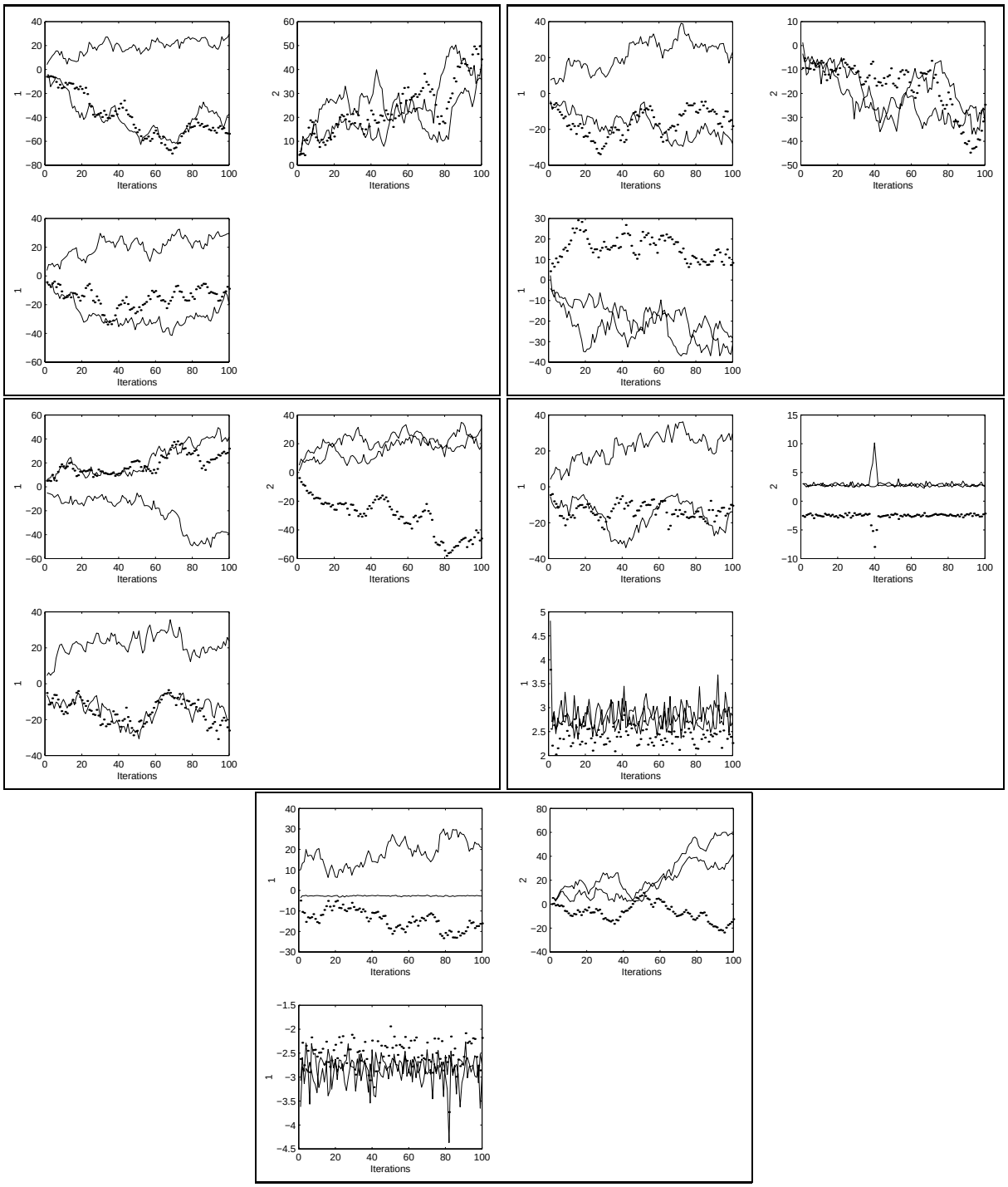

Figure 5.34: Evolution of mean values in the Fink model for an XOR experiment. Each of the five major subgraphs represent a single kernel in the probability model. Minor subgraphs are arranged and labeled by network unit number, with the vertical axis representing the mean value for a given variable.

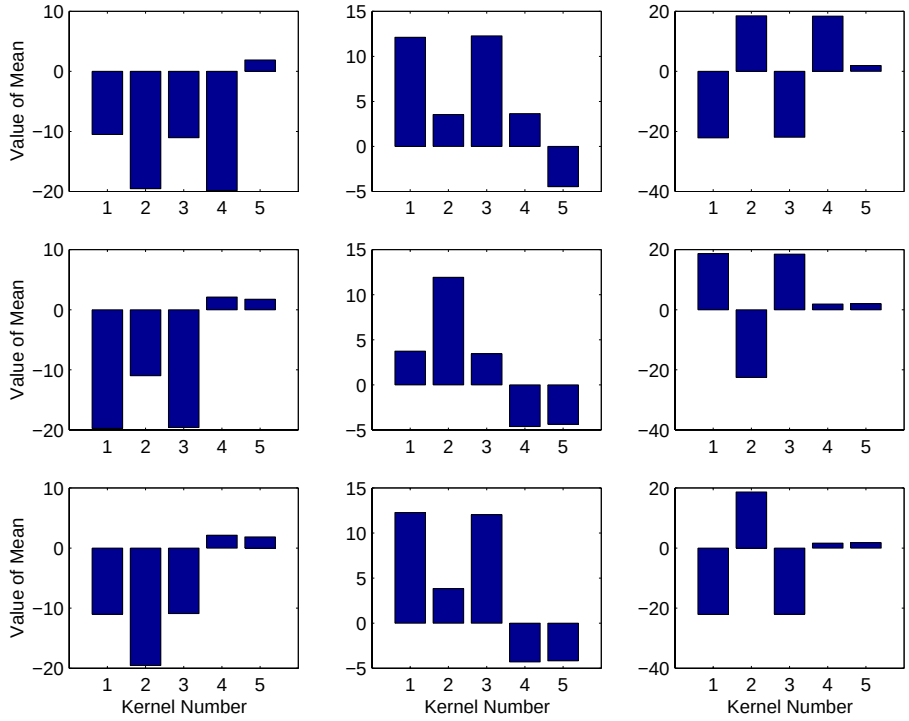

Figure 5.35: Bar chart representation of Fink final model for a XOR experiment. Mean values give an indication of the location of the model in the search space, and kernel number can be used to visually compare different kernels in the probability model.

For the Fink algorithm, the data describing the kernel density model remains of high dimensionality, especially when a large number of kernels is considered. Despite this difficulty, it is still possible to interpret the model at a given point in time, such as at the end of training. The reason this is possible is because the model is represented typically as a very *small number of points* in the high-dimensional weight space. In these experiments, five kernels corresponds to five points on the error surface. Figure 5.35 shows the model at the end of a training run on the XOR problem. Each bar chart corresponds to a different weight in the 2-2-1 network. Grouping the values in this way facilitates the interpretation of the different values obtained for each weight by each kernel. While the complete model evolution of the mean values (Figure 5.34) may become very crowded for larger networks, Figure 5.35 retains its interpretability due to the small number of effective points used to describe the model.

5.5 Summary

This chapter explores a class of global, stochastic, population-based optimization algorithms for training MLP's. Such algorithms are closely related to the motivations of previous chapters,

namely the visualization and analysis of the structure of the error surface, as the main idea of this class of algorithms is to construct and evolve a probabilistic model of the search space, which is used to perform optimization.

An algorithmic framework has been constructed for the description of this class of optimization algorithms. This framework is general enough to include existing stochastic and evolutionary algorithms as instantiations. More importantly, it shows how a large number of unsupervised learning methods (e.g. probability density estimation) can be incorporated into an algorithm of this kind. Two algorithms, based on Gaussian mixture model and kernel density estimation, have been proposed and implemented through this framework. These density estimators in an optimization context allow considerably more powerful models to be explicitly constructed as part of the search process.

Applications are used to demonstrate the dynamics of these algorithms. One benefit of realizing and developing algorithms in this framework is that the algorithm can be analyzed through the visualization of the probabilistic model. Because this model is explicit, this information is readily available for visualization. It also summarizes the progress of the algorithm, reducing the amount of information that must be handled by the visualization process. This method provides dynamic information regarding the structure of the error surface encountered by the learning process, and is an innovative method for the scientific visualization of optimization algorithms.

Chapter 6

Conclusion

6.1 Summary and Conclusions

The focus of this thesis has been the error surface of Multi-Layer Perceptrons (MLP's). Scientific visualization, statistical measurement and modelling through optimization algorithms have been the techniques used to investigate and better understand the structure of the error surface. This investigation has resulted in a methodology which is general enough to be applicable in practical MLP training situations, as well as being adaptable to a wider range of problems which involve high-dimensional optimization or configuration of large numbers of adaptive parameters.

Chapter 2 provided the background of the MLP network, and introduced the error surface as the solution space of the optimization problem that corresponds to the MLP training problem when performing supervised learning. The field of MLP research is discussed, in particular the very large amount of work which has been concerned with the design of more effective and efficient MLP training algorithms. It is shown that several fundamental difficulties remain in this area. The No Free Lunch (NFL) theorems serve as a reminder that in the absence of any knowledge particular to the optimization problem in question, all algorithms can be expected to perform equally well. Thus, the incorporation of heuristics, prior knowledge and assumptions into the training problem are seen as fundamental. Although these factors are in fact present in every practical optimization problem, they are quite often implicit, and their effects on the solution space, and hence on the expected performance of training algorithms, is unclear. The MLP training problem is a classic example of such an optimization problem. In addition to these

difficulties, many proposed algorithms do not contain adequate comparative studies, and results may be difficult to reproduce due to certain assumptions not being made explicit. Although some guidelines have been developed and effective algorithms suggested, it is difficult to choose an algorithm and methodology in a given situation, because this choice is problem dependent.

In Chapter 3, scientific visualization methods are considered to investigate the structure of the error surface and the dynamics of learning algorithms. It is shown that previous work on the application of visualization to ANN research (in particular MLP's) has been quite limited. The major difficulty that the error surface poses for effective visualization is seen as its typically very high dimensionality. Principal Component Analysis (PCA) is proposed as a technique for visualization of the data produced by learning trajectories of algorithms on the error surface. Results show that this data can be well represented through visualization of the first few principal components (PC's). In addition, the evolution of the principal values (PV's) in this case provides a picture of the relative contributions of each weight to the total variance of the learning trajectory as a function of training iterations. This visualization technique is applicable to networks of practical sizes, and is relatively inexpensive in storage and computational requirements. It makes no restrictive assumptions on the particular training situation, and is seen as a potentially very useful technique for comparing different learning algorithms on practical training tasks.

Chapter 4 provides a review of existing results and knowledge of the structure of MLP error surfaces. Typical features of the error surface are a large degree of symmetry, wide flat plateaus and narrow ravines due to ill-conditioning, and a possibly small number of true local minima. Statistical sampling techniques are proposed as a method for further exploratory analysis of this structure. Error histograms give an idea of the global distribution of error values on the surface. The shape of this distribution is a useful indication of the (global) difficulty of the surface for training algorithms. An alternative to sampling uniformly in some feasible region of the error surface is to restrict attention to a sample of points which match some criteria. Low-lying regions and apparent minima are examined in this chapter. To examine how such points are distributed about the error surface, the pairwise (histogram of distances between pairs of points in the sample) distribution is applied to several test problems. These results can be compared with a similar histogram of randomly sampled points, to reveal features such as clustering or multimodality in the locations of low-lying or apparent minima (AM) sample

points. The relationship between error and distance from a global minimum of the error surface can be investigated using the fitness-distance correlation (FDC) from evolutionary computation, with the restriction that the location of the global minimum is known *a priori*. FDC is calculated experimentally in the student-teacher learning paradigm, and the scatterplots reveal key features of the error surface, such as error plateaus, globally bowl-like regions and transitions in error values as a function of distance. Finally, ultrametric structure is detected in low-lying and AM samples on MLP error surfaces. This structure is found by calculating third-order statistics (distances between points in randomly selected triangles) of the samples. This result is an addition to the other examples of complex solution spaces from combinatorial optimization and statistical physics which also display a high degree of ultrametricity. In contrast however, the results of this chapter are for a continuous search space. This establishes a connection between these different solution spaces, which requires further investigation and explanation. Any widely occurring structural feature of search spaces should be eventually exploitable by algorithms, thus leading to improved performance.

Chapter 5 considered global, stochastic, population-based optimization algorithms as maintaining a model of the structure of the error surface during the optimization process. A general algorithmic framework which meets this requirement is developed. It is shown through the application of this framework that several relevant existing optimization algorithms can be viewed as particular instantiations of this framework. This insight changes the perspective of how these algorithms operate. New algorithms of this class are then developed using an adaptive Gaussian mixture model and finite kernel probability density estimation methods. These techniques are capable of constructing more sophisticated models of the error surface, such as multimodality. The algorithm based on finite kernels can be seen as a natural extension of existing EA's from the perspective of the framework developed. The search behaviour of these algorithms can be viewed as a parallel, stochastic, coupled search procedure using a population of solutions. The algorithm based on Gaussian kernels is applied to MLP training problems. It is shown that an additional benefit of the algorithm and its framework is the potential for visualization of the evolution of the probabilistic model of the error surface, during training, and the model which is obtained after training.

Overall, these chapters show that understanding the structure of the error surface leads to a better understanding of the MLP training problem. Furthermore, this structure can be incorpo-

rated into the training process in a simple way, which allows the inclusion of prior knowledge, heuristics and a wide variety of probabilistic modelling techniques. Hence, the structure of the error surface should be an important consideration in the understanding of MLP training algorithms, leading to the design of better algorithms.

6.2 Limitations and Suggestions for Future Work

This thesis is an investigation into techniques for exploratory analysis of MLP error surfaces. Ideas from a number of disparate fields have been drawn together for this investigation, and it is clear that analysis of the configuration space in complex systems in general is an area which contains much scope for development and elaboration. A number of directions for further research are suggested by the work of in this thesis:

- *High-dimensional Optimization.* It remains unclear if there are any universal characteristics of the configuration spaces of high-dimensional optimization problems (and other such systems). It is known that high-dimensional spaces have some unusual properties, but it is not clear if results such as the central limit catastrophe and the extra-dimensional bypass have any consequences for practical problem instances. Interestingly, Figures 4.13- 4.15 do not show as high error values as other figures, leaving open the possibility that the above-mentioned central limit catastrophe can be observed in MLP error surfaces. Verification of this possibility requires further experimentation.

- *Sampling Statistics on Error Surfaces.* While the statistical properties of error surfaces used in this thesis are useful for exploratory analysis, it would be interesting to conduct a large empirical study of the ability of these methods to predict the performance of different algorithms, or even the sensitivity of parameter settings in algorithms to variation in the error surface.

- *Practical Implications of Ultrametric Structure.* Ultrametricity in spin glass systems seems to be a property of frustrated, disordered systems (frustrated in the sense that there is competition among conflicting interactions - the system does not find one accommodation that satisfactorily satisfies all constraints; disordered in that there are an infinite number of parameters with nontrivial complexity). Further work is required to see if a

similar relationship can be found in MLP error surfaces, with the variation of factors such as the nature of the training set and the topology of the network. It also remains to be shown if information such as ultrametricity can be put to use in an efficient manner in an algorithm.

- *Probabilistic Global Optimization.* This thesis has considered only preliminary instantiations of the framework of Chapter 5. Practical algorithms require further work, particularly to investigate the possibility of adapting algorithmic parameters automatically as part of the optimization process. There has been much interest recently in the modelling of statistical dependencies between a number of variables using *graphical models* (e.g. [194]). It may be possible to incorporate techniques from this field into optimization algorithms. In some ways this direction has been taken with extending the discrete-PBIL algorithm [20, 60]. It would also be interesting to explore methods of density estimation which are especially suited to modelling the distribution of non-stationary data in an on-line situation.

  Different schemes could be explored for maintaining and modifying the kernels in the Fink algorithm. For example, each kernel could be assigned a finite "life span" which is decremented after each iteration and replenished only when that particular kernel produces the global best sample in the current iteration. Redundant kernels could then be removed, and replaced with new ones. This modification implements a kind of crude memory mechanism into the model.

Bibliography

[1] L. Altenberg. Fitness distance correlation analysis: An instructive counterexample. In T. Baeck, editor, *Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 57–64, San Francisco, CA, 1997. Morgan Kauffman.

[2] R. W. Anderson. Biased random-walk learning: a neurobiological correlate to trial-and-error. Technical report, Smith-Kettlewell Eye Research Institute, 1993.

[3] R. Andrews, J. Diederich, and A. B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, 1995.

[4] G. S. Androulakis, G. D. Magoulas, and M. N. Vrahatis. Geometry of learning: Visualizing the performance of neural network supervised training methods. *Nonlinear Analysis, Theory, Methods & Applications*, 30(7):4359–4544, 1997.

[5] M. A. Arbib. Part I: Background. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1–25. MIT Press, Cambridge, MA, 1995.

[6] M. A. Arbib. Part II: Road maps. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 27–58. MIT Press, Cambridge, MA, 1995.

[7] P. Auer, M. Herbster, and M. K. Warmuth. Exponentially many local minima for single neurons. In D. Touretzky, M. Mozer, and M. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 316–322, Cambridge, MA, 1996. The MIT Press.

[8] N. Baba. A new approach for finding the global minimum of error function of neural networks. *Neural Networks*, 2(5):367–374, 1989.

[9] N. Baba, Y. Mogami, M. Kohzaki, Y. Shiraishi, and Y. Yoshida. A hybrid algorithm for finding the global minimum of error function of neural networks and its applications. *Neural Networks*, 7(8):1253–1265, 1994.

[10] S. Bacci and N. Parga. Ultrametricity, frustration and the graph colouring problem. *Journal of Physics A*, 22:3023–3032, 1989.

[11] T. Bäck and H-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1:1–23, 1993.

[12] P. Bakker, S. Phillips, and J. Wiles. The N-2-N encoder: A matter of representation. In S. Gielen and B. Kappen, editors, *International Conference on Artifical Neural Networks*, pages 554–557, Berlin, 1993. Springer-Verlag.

[13] P. Bakker, S. Phillips, and J. Wiles. The 1000-2-1000 encoder: A matter of representation. *Neural Network World*, 4(5):527–534, 1994.

[14] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[15] P. Baldi and K. Hornik. Learning in linear networks: a survey. *IEEE Transactions on Neural Networks*, 6(4):837–858, 1995.

[16] S. Baluja. Population-Based Incremental Learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, School of Computer Science, Carnegie Mellon University, 1994.

[17] S. Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Carnegie Mellon University, September 1995.

[18] S. Baluja. Genetic algorithms and explicit search statistics. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 319–325, Cambridge, MA, 1997. The MIT Press.

[19] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. Technical Report CMU-CS-95-141, Carnegie Mellon University, May 1995.

[20] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. Technical Report CMU-CS-97-107, Carnegie Mellon University, January 1997.

[21] J. Barhen, A. Fijany, and N. Toomarian. Globally optimal neural learning. In *World Congress on Neural Networks*, volume III, pages 370–375, Hillsdale, NJ, 1994. Lawrence Erlbaum Associates.

[22] E. Barnard. Optimization for training neural nets. *IEEE Transactions on Neural Networks*, 3(2):232–240, 1992.

[23] E. Barnard and D. Casasent. A comparison between criterion functions for linear classifiers, with an application to neural nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1030–1041, 1989.

[24] A. G. Barto. Learning as hill-climbing in weight space. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 531–533. The MIT Press, Cambridge, MA, 1995.

[25] R. Battiti. Accelerated backpropagation learning: Two optimization methods. *Complex Systems*, 3:331–342, 1989.

[26] R. Battiti. First- and second-order methods for learning: between steepest descent and Newton's method. *Neural Computation*, 4:141–166, 1992.

[27] R. Battiti and G. Tecchiolli. Training neural nets with the reactive tabu search. *IEEE Transactions on Neural Networks*, 6(5):1185–1200, 1995.

[28] J. E. Beasley. Population heuristics. Retrieved from http://mscmga.ms.ic.ac.uk/jeb/jeb.html (1/12/99), March 1999. Submitted.

[29] B. P. Bergeron. Using a spreadsheet metaphor to visualize network behaviour. *Collegiate-Microcomputer*, 8(2):81–92, 1990.

[30] M. Betrouni, S. Delsert, and D. Hamad. Interactive pattern classification by means of artificial neural networks. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3275–3279, 1995.

[31] M. Bianchini, P. Frasconi, and M. Gori. Learning in multilayered networks used as autoassociators. *IEEE Transactions on Neural Networks*, 6(2):512–515, 1995.

[32] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, 1995.

[33] C. M. Bishop and M. E. Tipping. A hierarchical latent variable model for data visualization. Technical Report NCRG-96-028, Neural Computing Research Group, Aston University, 1998.

[34] C. Blake, E. Keogh, and C.J. Merz. UCI repository of machine learning databases. Retrieved from http://www.ics.uci.edu/∼mlearn/MLRepository.html (1/12/99), 1998.

[35] E. K. Blum. Approximation of boolean functions by sigmoidal networks: Part I: XOR and other two-variable functions. *Neural Computation*, 1:532–540, 1989.

[36] K. D. Boese. *Models for Iterative Global Optimization*. PhD thesis, University of California, Los Angeles, 1996.

[37] K. D. Boese and A. B. Kahng. Simulated annealing of neural networks: the ”cooling” strategy reconsidered. In *IEEE International Symposium on Circuits and Systems*, volume 4, pages 2572–2575, New York, NY, 1993. IEEE.

[38] J. P. Bouchaud and P. Le Doussal. Ultrametricity transition in the graph colouring problem. *Europhysics Letters*, 1(3):91–98, 1986.

[39] J. Branke. Evolutionary algorithms for neural network design and training. Technical Report 332, Institute AIFB, University of Karlsruhe, Germany, January 1995.

[40] J. M. Chambers, W. S. Cleveland, B. Kleiner, and P. A. Tukey. *Graphical Methods for Data Anlaysis*. Wadsworth International, Boston, MA, 1983.

[41] A. M. Chen, H. Lu, and R. Hecht-Nielsen. On the geometry of feedforward neural network error surfaces. *Neural Computation*, 5(6):910–927, 1993.

[42] C-H Chen, R. G. Parekh, J. Yang, K. Balakrishnan, and V. Honavar. Analysis of decision boundaries generated by constructive neural network learning algorithms. In *World*

Congress on Neural Networks, volume 1, pages 628–635, Mahwah, NJ, 1995. Lawrence Erlbaum Associates.

[43] C. O-T. Chen and B. J. Sheu. Optimization schemes for neural network training. In *IEEE International Conference on Neural Networks*, pages 817–822, Piscataway, NJ, 1994. IEEE.

[44] D. L. Chester. Why two hidden layers are better than one. In *International Joint Conference on Neural Networks*, volume I, pages 265–268. Lawrence Erlbaum, 1990.

[45] Y. J. Choie, S. Kin, and C. N. Lee. Chaotic dynamics and the geometry of the error surface in neural networks. *Physica D*, 55:113–120, 1992.

[46] W. S. Cleveland. *Visualizing Data*. Hobart Press, Summit, New Jersey, 1993.

[47] W. S. Cleveland. *The Elements of Graphing Data*. Hobart Press, Summit, New Jersey, 1994.

[48] H. G. Cobb. Is the genetic algorithm a cooperative learner? In D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 277–296. Morgan Kaufmann, 1993.

[49] F. M. Coetzee and V. L. Stonick. 488 solutions to the XOR problem. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, volume 9, pages 410–416, Cambridge, MA, 1997. The MIT Press.

[50] E. Colet and D. Aaronson. Visualization of multivariate data: Human-factors considerations. *Behaviour Research Methods, Instruments & Computers*, 27(2):257–263, 1995.

[51] comp.ai.neural-nets Usenet Newsgroup FAQ. Retrieved from ftp://ftp.sas.com/pub/neural/FAQ.html (1/12/99).

[52] comp.ai.genetic Usenet Newsgroup FAQ (Hitchhiker's guide to evolutionary computation). Retrieved from ftp://rtfm.mit.edu/pub/usenet/news.answers/ai-faq/genetic/ (2/12/99).

[53] M. Conrad. The geometry of evolution. *BioSystems*, 24:61–81, 1990.

[54] M. Conrad and W. Ebeling. M. V. Volkenstein, evolutionary thinking and the structure of fitness landscapes. *BioSystems*, 27:125–128, 1992.

[55] R. Crane, C. Fefferman, S. Markel, and J. Pearson. Characterizing neural network error surfaces with a sequential programming algorithm. In *Machines That Learn*, Snowbird, 1995.

[56] M. W. Craven and J. W. Shavlik. Visualizing learning and computation in artificial neural networks. Technical Report 91-5, University of Wisconsin Computer Sciences Department, 1991.

[57] Y. Le Cun, I. Kanter, and S. A. Solla. Second order properties of error surfaces: Learning time and generalization. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 918–924, San Mateo, CA, 1991. Morgan Kaufmann.

[58] C. J. Darken. Stochastic approximation and neural network learning. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 941–945. MIT Press, Cambridge, MA, 1995.

[59] J. Darr. Back propagation family album. Technical Report TR96-05, Department of Computing, Macquarie University, August 1996.

[60] J. S. De Bonet, C. L. Isbell, Jr., and P. Viola. MIMIC: Finding optima by estimating probability densities. In *Advances in Neural Information Processing Systems*, volume 9, pages 424–430, 1997.

[61] K. A. De Jong. Genetic algorithms are NOT function optimizers. In D. Whitley, editor, *Foundations of Genetic Algorithms*, volume 2, pages 5–17, San Mateo, CA, 1993. Morgan Kaufmann.

[62] J. de Villiers and E. Barnard. Backpropagation neural nets with one and two hidden layers. *IEEE Transactions on Neural Networks*, 4(1):136–141, 1992.

[63] J. Denker, D. Schwartz, B. Wittner, S. Solla, R. Howard, and L. Jackel. Large automatic learning, rule extraction and generalization. *Complex Systems*, 1:877–922, 1987.

[64] S. Dennis and S. Philips. Analysis tools for neural networks. Technical Report 207, University of Queensland Key Center for Software Technology, May 1991.

[65] K. I. Diamantaras and S. Y. Kung. *Principal component neural networks theory and applications*. John Wiley and Sons, New York, 1996.

[66] T. G. Dietterich. Approximate statistical tests for compering supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1996.

[67] L. C. W. Dixon, J. Gomulka, and S. E. Hersom. Reflections on the global optimization problem. In L. C. W. Dixon, editor, *Optimization in Action*, pages 398–435. Academic Press, 1975.

[68] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on systems, Man and Cybernetics - Part B*, 26(1):1–13, 1996. (see http://iridia.ulb.ac.be/dorigo/ACO/ACO.html).

[69] J. L. Elman. Representation and structure in connectionist models. In G. T. Altmann, editor, *Cognitive Models of Speech Processing*, chapter 17, pages 345–382. MIT Press, Cambridge, MA, 1990.

[70] J. L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–224, 1991.

[71] S. Ergezinger. An accelerated learning algorithm for multilayer perceptrons: Optimization layer by layer. *IEEE Transactions on Neural Networks*, 6(1):31–42, 1995.

[72] B. S. Everitt. *Graphical Techniques for Multivariate Data*. Heinemann Educational Books, London, 1978.

[73] B. S. Everitt. *Cluster Analysis*. Edward Arnold, London, UK, 1993.

[74] S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 Connectionist Models Summer School*, pages 38–51, Carnegie-Mellon University, 1988.

[75] R. Fletcher. *Practical methods of optimization*. Wiley, Chichester, New York, 2nd edition, 1987.

[76] D. B. Fogel. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1994.

[77] E. Galić and M. Höhfeld. Improving the generalization performance of multi-layer-perceptrons with population-based incremental learning. In *Parallel Problem Solving from Nature (PPSN IV)*, Lecture Notes in Computer Science (vol. 1141), pages 740–750, Berlin, New York, 1996. Springer.

[78] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilema. *Neural Computation*, 4(1):1–58, 1992.

[79] G. J. Gibson and C. F. N. Cowan. On the decision regions of multi-layer perceptrons. *Proceedings of the IEEE*, 78(10):1590–1594, 1990.

[80] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.

[81] M. Gori and M. Maggini. Optimal convergence of on-line backpropagation. *IEEE Transactions on Neural Networks*, 7(1):251–254, 1996.

[82] M. Gori and A. Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-14:76–86, 1992.

[83] M. Gori and A. C. Tsoi. Comments on local minima free conditions in multilayer perceptrons. *IEEE Transactions on Neural Networks*, 9(5):1051–1053, 1998.

[84] D. Gorse, A. J. Shepherd, and J. G. Taylor. The new ERA in supervised learning. *Neural Networks*, 10(2):343, 1997.

[85] E. Gullichsen and E. Chang. Pattern classification by neural network: An experimental system for icon recognition. In M. Caudill and C. Butler, editors, *1st International Conference on Neural Networks*, volume 4, pages 725–732, San Diego, CA, 1987. IEEE.

[86] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 5:989–993, 1994.

[87] L. G. C. Hamey. The structure of neural network error surfaces. In M. Charles and C. Latimer, editors, *Proc. Sixth Australian Conference on Neural Networks*, pages 197–200. University of Sydney, 1995.

[88] L. G. C. Hamey. Analysis of the error surface of the xor network with two hidden nodes. In P. Bartlett, A. Burkitt, and R. Williamson, editors, *Seventh Australian Conference on Neural Networks (ACNN'96)*, pages 179–183, Canberra, Australia, 1996. Australian National University.

[89] L. G. C. Hamey. XOR has no local minima: A case study in neural network error surface analysis. *Neural Networks*, 11(4):669–681, 1998.

[90] H. M. Hastings. The May-Wigner stability theorem. *Journal of Theoretical Biology*, 97:155–166, 1982.

[91] S. Haykin. *Neural Networks A Comprehensive Foundation*. Prentice-Hall, Upper Saddle River, NJ, second edition, 1999.

[92] R. Hecht-Nielsen. Theory of backpropagation neural network. In *International Joint Conference on Neural Networks*, pages 593–605, San Diego, CA, 1989. IEEE, IEEE.

[93] R. Hecht-Nielsen. *Neurocomputing*. Addison-Wesley, Reading, MA, 1990.

[94] R. Hecht-Nielsen. On the algebraic structure of feedforward neural network weight spaces. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 129–136. Elsevier Science Publishers B.V, 1990.

[95] R. Hecht-Nielsen. The munificence of high dimensionality. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks 2*, pages 1017–1030, Amsterdam, New York, 1992. North-Holland.

[96] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*, volume I of *Santa Fe Institute studies in the sciences of complexity*. Addison-Wesley, Redwood City, CA, 1991.

[97] G. E. Hinton, J. L. McClelland, and D. E. Rumelhart. Distributed representations. In *Parallel Distributed Processing*, volume 1, chapter 3, pages 77–109. MIT Press, Cambridge, MA, 1986.

[98] Y. Chong Ho and J. T. Behrens. Applications of multivariate visualization to behavioural sciences. *Behaviour Research Methods, Instruments & Computers*, 27(2):264–271, 1995.

[99] S. Hochreiter and J. Schmidhuber. Flat minimum search finds simple nets. Technical Report FKI-200-94, Technische Universität München, December 1994.

[100] S. Hochreiter and J. Schmidhuber. Simplifying neural nets by discovering flat minima. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 529–536, Cambridge, MA, 1995. The MIT Press.

[101] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[102] T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. In *National Conference on Artificial Intelligence (AAAI-93)*, pages 231–236, Menlo Park, CA, 1993. AAAI.

[103] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 2nd edition, 1992.

[104] W. Hordijk. A measure of landscapes. Technical Report SFI-TR-95-05-049, Santa-Fe Institute, 1995.

[105] W. Hordijk and P. F. Stadler. Amplitude spectra of fitness landscapes. Technical Report SFI-TR-98-02-021, Santa-Fe Institute, 1998.

[106] Y. Horikawa. Landscapes of basins of local minima in the XOR problem. In *International Joint Conference on Neural Networks*, volume 2, pages 1677–1680, New York, 1993. IEEE.

[107] J. Hu, K. Hirasawa, J. Murata, M. Ohbayashi, and Y. Eki. A new random search method for neural network learning - RasID. In *International Joint Conference on Neural Networks*, pages 2346–2351, 1998.

[108] S. J. Huang, S. N. Koh, and H. K. Tang. Training algorithm based on Newton's method with dynamic error control. In *International Joint Conference on Neural Networks*, volume III, pages 899–904, New York, 1992. IEEE.

[109] B. A. Huberman. The performance of cooperative processes. *Physica D*, 42:38–47, 1990.

[110] D. R. Hush, B. Horne, and J. M. Salas. Error surfaces for multi-layer perceptrons. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1152–1161, 1992.

[111] D. R. Hush, J. M. Salas, and B. Horne. Error surfaces for multi-layer perceptrons. In *International Joint Conference on Neural Networks (Seattle)*, volume I, pages 759–764, New York, 1991. IEEE.

[112] E. J. Jackson. *A User's Guide to Principal Components*. Wiley, New York, 1991.

[113] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[114] R. A. Jarvis. Adaptive global search by the process of competitive evolution. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-5(3):297–311, 1975.

[115] T. T. Jervis and W. J. Fitzgerald. Optimization schemes for neural networks. Technical Report TR 144, Cambridge University Engineering Department, England, August 1993.

[116] L. O. Jimenez and D. A. Landgrebe. Supervised classification in high-dimensional space: Geometrical, statistical, and asymptotical properties of multivariate data. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 28(1):39–54, 1998.

[117] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

[118] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, 1995.

[119] W. T. Jones, R. K. Vachha, and A. P. Kulshrestha. DENDRITE: A system for the visual interpretation of neural network data. In *SOUTHEASTCON'92*, volume 2, pages 638–641, New York, 1992. IEEE.

[120] F. Jordan and G. Clement. Using the symmetries of a multi-layered network to reduce the weight space. In *International Joint Conference on Neural Networks (Seattle)*, volume II, pages 391–396, New York, 1991. IEEE.

[121] A. B. Kahng. Exploiting fractalness of error surfaces: new methods for neural network learning. In *IEEE International Symposium on Circuits and Systems*, volume 1, pages 41–4, New York, 1992. IEEE.

[122] P. P. Kanjalil and D. N. Banerjee. On the application of orthogonal transformation for the design and analysis of feedforward networks. *IEEE Transactions on Neural Networks*, 6(5):1061–1070, 1995.

[123] D. A. Karras and S. J. Perantonis. An efficient constrained training algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 6(6):1420–1434, 1995.

[124] S. Kauffman. *The Origins of Order*, chapter 2, pages 33–67. Oxford University Press, Oxford, 1993.

[125] S. Kauffman. *At Home in the Universe*, chapter 11, pages 245–271. Oxford University Press, Oxford, 1995.

[126] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, pages 1942–1948, New York, 1995. IEEE.

[127] S. Khuri and J. Williams. Neuralis: an artificial neural network package. *SIGCSE-Bulletin*, 28:25–27, 1996.

[128] J. Kindermann and A. Linden. Inversion of neural networks by gradient descent. *Parallel Computing*, 14:277–286, 1990.

[129] S. Kirkpatrick and G. Toulouse. Configuration space analysis of travelling salesman problems. *Journal de Physique (Paris)*, 46:1277–1292, 1985.

[130] J. Kolen and J. Pollack. Back propagation is sensitive to initial conditions. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 860–867, San Mateo, CA, 1991. Morgan Kaufmann.

[131] J. F. Kolen and A. K. Goel. Learning in parallel distributed processing networks: Computational complexity and information content. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(2):359–367, 1991.

[132] L. Kruglyak. How to solve the N bit encoder problem with just two hidden units. *Neural Computation*, 2(4):399–401, 1990.

[133] W. J. Krzanowski. *Principles of Multivariate Analysis: A User's Perspective.* Clarendon Press, Oxford, 1988.

[134] V. Kůrková and P. Kainen. Functionally equivalent feedforward neural networks. *Neural Computation*, 6(3):543–558, 1994.

[135] V. Kvasnicka, M. Pelikan, and J. Pospichal. Hill climbing with learning (an abstraction of genetic algorithm). In *First Online Workshop on Evolutionary Computation*, http://www.bioele.nuee.nagoya-u.ac.jp/wec/ (as at 3/12/99), 1995.

[136] K. J. Lang and M. J. Witbrock. Learning to tell two spirals apart. In T. Sejnowski D. Touretzky, G. Hinton, editor, *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59, San Mateo, CA, 1988. Morgan Kaufmann.

[137] S. R. Lawrence. *Neural Networks For Real World Tasks Limitations and Solutions*. PhD thesis, University of Queensland, 1997.

[138] Y. LeCun, L. Bottou, G. B. Orr, and K-R. Müller. Efficient backprop. In G. B. Orr and K-R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, chapter 1, pages 9–50. Springer, 1998.

[139] M. Lehr. *Scaled Stochastic Methods for Training Neural Networks*. PhD thesis, Stanford University, January 1996.

[140] A. Levin, T. Leen, and J. Moody. Fast learning using principal components. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 35–42, San Francisco, CA, 1994. Morgan Kaufmann.

[141] Y. Liao and J. Moody. A neural network visualization and sensitivity analysis toolkit. In S. Amari et. al., editor, *International Conference on Neural Information Processing (ICONIP'96)*, pages 1069–1074, Berlin, New York, 1996. Springer.

[142] P. J. G. Lisboa and S. J. Perantonis. Complete solution of the local minima in the XOR problem. *Network*, 2:119–124, 1991.

[143] R. Lister. Back propagation and the N-2-N encoder. In P. Leong and M. Jabri, editors, *Australian Conference on Neural Networks (ACNN'92)*, pages 198–201, Australia, 1992. University of Sydney.

[144] R. Lister. Visualizing weight dynamics in the N-2-N encoder. In *IEEE International Conference on Neural Networks*, volume 2, pages 684–689, Piscataway, NJ, 1993. IEEE.

[145] R. Lister. Fractal strategies for neural network scaling. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 403–405. MIT Press, 1995.

[146] R. Maclin and J. W. Shavlik. Combining the predictions of multiple classifiers: using competitive learning to initialize neural networks. In *IJCAI-95. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 524–530, USA, 1995.

[147] W. Macready and D. Wolpert. What makes an optimization problem hard? Technical Report SFI-TR-95-05-046, The Santa Fe Institute, February 1996.

[148] H. A. Malki and A. Moghaddamjoo. Using the Karhunen-Loe'ye transformation in the back-propagation training algorithm. *IEEE Transactions on Neural Networks*, 2(1):162–165, 1991.

[149] B. F. J. Manly. *Multivariate Statistical Methods A Primer*. Chapman and Hall, London, 1994.

[150] J. J. McKeown, F. Stella, and G. Hall. Some numerical aspects of the training problem for feed-forward neural nets. *Neural Networks*, 10(8):1455–1463, 1997.

[151] D. A. Medler. A brief history of connectionism. *Neural Computing Surveys*, 1:61–101, 1998. http://www.icsi.berkeley.edu/˜jagota/NCS (as at 3/12/99).

[152] M. L. Minksy and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, Cambridge, MA, expanded edition, 1988.

[153] M. F. Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 1993.

[154] G. J. Mpitsos and R. M. Burton, Jr. Convergence and divergence in neural networks: Processing of chaos and biological analogy. *Neural Networks*, 5(4):605–625, 1992.

[155] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[156] S. Mukherjee. *Neural Network Training Algorithms Based on Quadratic Error Surface Models*. PhD thesis, Cornell University, 1997.

[157] B. Müller, J. Reinhardt, and M. T. Strickland. *Neural Networks An Introduction*. Physics of Neural Networks. Springer-Verlag, Berlin, second edition, 1995.

[158] P. W. Munro. Visualizations of 2-D hidden unit space. In *International Joint Conference on Neural Networks*, volume III, pages 468–473, New York, 1992. IEEE.

[159] T. Nabhan and A. Zomaya. Toward generating neural network structures for function approximation. *Neural Networks*, 7(1):89–100, 1994.

[160] R. M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer-Verlag, New York, 1996.

[161] R. M. Neal. Assessing relevance determination methods using DELVE. In C. M. Bishop, editor, *Generalization in Neural Networks and Machine Learning*, pages 97–129. Springer-Verlag, Berlin, 1998.

[162] N. J. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1965.

[163] E. Oja. Principal component analysis. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 753–756. MIT Press, Cambridge, MA, 1995.

[164] M. Opper and W. Kinzel. Statistical mechanics of generalization. In E. Domany, J. L. van Hemmen, and K. Schulten, editors, *Models of Neural Networks III*, chapter 5, pages 151–209. Springer-Verlag, New York, 1996.

[165] C. Ornes and J. Sklansky. A neural networks that visualizes what it classifies. *Pattern Recognition Letters*, 18:1301–1306, 1997.

[166] G. B. Orr and T. K. Leen. Weight space probability densities in stochastic learning: II. transients and basin hopping times. In S. Hanson, J. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 507–514, San Mateo, CA, 1993. Morgan Kaufmann.

[167] G. B. Orr and K-R. Müller, editors. *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[168] B. Orsier. Another hybrid algorithm for finding a global minimum of MLP error functions. Technical Report UNIGE-AI-95-6, CUI, University of Geneva, January 1996.

[169] A. Ossen and S. M. Rüller. An analysis of the metric structure of the weight space of feedforward networks and its application to time series modeling and prediction. In M. Verleysen, editor, *4th European Symposium on Artificial Neural Networks (ESANN'96)*, pages 315–322, Brussels, Belgium, 1996. D Facto.

[170] *Proceedings of the Parallel Problem Solving from Nature Conference*, Berlin, 1993 - 1999. Springer-Verlag.

[171] J. Paredis. Coevolutionary algorithms. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *The Handbook of Evolutionary Computation*. Oxford University Press, Oxford, 1997.

[172] R. Parisi, E. Di Claudio, and G. Orlandi. A generalized learning paradigm exploiting the structure of feedforward neural networks. *IEEE Transactions on Neural Networks*, 7(6):1450–1459, 1996.

[173] D. Partridge. Network generalization differences quantified. *Neural Networks*, 9(2):263–271, 1996.

[174] M. Peyral, A. Docoulombier, C. Ravise, M. Schoenauer, and M. Sebag. Mimetic evolution. In J-K. Hao et. al., editor, *3rd European Conference on Artifical Evolution (AE'97)*, volume 1363 of *Lecture Notes in Computer Science*, pages 81–94, Berlin, New York, 1997. Springer.

[175] S. Phillips. The effect of representation on error surface. In P. Leong and M. Jabri, editors, *Fourth Australian Conference on Neural Networks (ACNN'93)*, pages 86–89, Australia, 1993. University of Sydney.

[176] A. J. Pinz and H. Bischof. Constructing a neural network for the interpretation of the species of trees in aerial photographs. In *10th International Conference on Pattern Recognition*, volume 1, pages 755–757, Los Alamitos, CA, 1990. IEEE Comp. Soc. Press.

[177] T. Plate, J. Bert, J. Grace, and P. Band. Visualizing the function computed by a feedforward neural network. Technical Report CS-TR-98-5, Victoria University of Wellington, July 1998.

[178] J. B. Pollack. Connectionism: Past, present, and future. *Artificial Intelligence Review*, 3:3–20, 1989.

[179] V. W. Porto. Alternative neural network training methods. *IEEE Expert*, 1995.

[180] T. Poston, C. Lee, Y. J. Choie, and Y. Kwon. Local minima and backpropagation. In *International Joint Conference on Neural Networks*, volume II, pages 173–176, New York, 1991. IEEE.

[181] L. Prechelt. Proben1 - a set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Universität Karlsruhe, Germany, September 1994.

[182] L. Prechelt. A study of experimental evaluations of neural network learning algorithms: Current research practice. Technical Report 19/94, Universität Karlsruhe, Germany, August 1994.

[183] L. Prechelt. Some notes on neural learning algorithm benchmarking. *Neurocomputing*, 9(3):343–347, 1995.

[184] L. Prechelt. Investigation of the CasCor family of learning algorithms. *Neural Networks*, 10(5):885–896, 1997.

[185] C. E. Priebe. Adaptive mixtures. *Journal of the American Statistical Association*, 89(427):796–806, 1994.

[186] W. Purgathofer and H. Loffelmann. Selected new trands in scientific visualization. *Proceedings of the SPIE*, 3346:130–145, 1998.

[187] D. Raghavarao. *Exploring Statistics*. Marcel Dekker, New York, 1988.

[188] R. Rammal, G. Toulouse, and M. A. Virasoro. Ultrametricity for physicists. *Reviews of Modern Physics*, 58(3):765–788, 1986.

[189] S. S. Rao. *Engineering optimization : theory and practice*. Wiley, New York, 3rd edition, 1996.

[190] R. Reed. Pruning algorithms - a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[191] R. G. Reynolds. An introduction to cultural algorithms. In A. Sebald and L. Fogel, editors, *3rd Annual Conference on Evolutionary Programming*, pages 131–139, Singapore, 1994. World Scientific.

[192] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons - From backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, 16:265–278, 1994.

[193] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591, Piscataway, NJ, 1993. IEEE.

[194] B. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.

[195] S. Rochet, G. Venturini, M. Slimane, and E. M. El Kharoubi. A critical and empirical study of epistasis measures for predicting GA performances: A summary. In J.-K. Hao et. al., editor, *3rd European Conference on Artificial Evolution (AE'97)*, volume 1363 of *Lecture Notes in Computer Science*, pages 275–285, Berlin, New York, 1997. Springer.

[196] R. Rojas. The boolean sphere: a geometrical approach to perceptron learning. In *IEEE International Conference on Neural Networks*, volume III, pages 358–363, Pascataway, NJ, 1994. IEEE.

[197] R. Rojas. The fractal geometry of backpropagation. In *Interational Conference on Neural Networks*, volume 1, pages 233–238, Piscataway, NJ, 1994. IEEE.

[198] R. Rojas. Oscillating iteration paths in neural networks learning. *Computers and Graphics*, 18(4):593–597, 1994.

[199] R. Rojas. Visualizing the learning process for neural networks. In M. Verleysen, editor, *European Symposium on Artificial Neural Networks (ESANN'94)*, pages 211–216, Brussels, Belgium, 1994. D Facto.

[200] B. E. Rosen and J. M. Goodwin. VFSR trained artificial neural networks. In *International Joint Conference on Neural Networks*, pages 2959–2962, New York, 1993. IEEE.

[201] S. Rudlof and M. Köppen. Stochastic hill climbing with learning by vectors of normal distributions. In *1st Online Workshop on Soft Computing*, Retrieved from http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/ (8/12/99), 1996.

[202] S. M. Rüger and A. Ossen. Clustering in weight space of feedforward nets. In C. et. al. von der Malsburg, editor, *International Conference on Artificial Neural Networks (ICANN'96)*, Lecture Notes in Computer Science 1112, pages 83–88, Berlin, 1996. Springer.

[203] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, chapter 8, pages 318–362. MIT Press, Cambridge, MA, 1986.

[204] S. Saarinen, R. Bramley, and G. Cybenko. Ill-conditioning in neural network training problems. *SIAM Journal of Scientific Computing*, 14(3):693–714, May 1993.

[205] R. Salomon. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *Biosystems*, 39:263–278, 1996.

[206] R. Salomon. Raising theoretical questions about the utility of genetic algorithms. In P. Angline et. al., editor, *Evolutionary Programming VI: 6th International Conference (EP'97)*, volume 1213 of *Lecture Notes in Computer Science*, pages 276–284, Berlin, New York, 1997. Springer.

[207] W. F. Schmidt, S. Raudys, M. A. Kraaijveld, M. Skurikhina, and R. P. W. Duin. Initialization, back-propagation and generalization of feed-forward classifiers. In *IEEE International Conference on Neural Networks*, pages 598–604, New York, 1993. IEEE.

[208] N. N. Schraudolph and T. J. Sejnowski. Tempering backpropagation networks: Not all weights are created equal. In G. Tesauro J. Cowan and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 563–569, San Francisco, CA, 1996. Morgan Kaufmann.

[209] H-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, New York, 1981.

[210] H-P. Schwefel. *Evolution and Optimum Seeking*. Wiley, New York, 1995.

[211] D. W. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization*. Wiley, New York, 1992.

[212] M. Sebag and A. Ducoulombier. Extending Population-Based Incremental Learning to continuous search spaces. In A. Eiben et. al., editor, *Parallel Problem Solving from Nature - PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 418–427, Berlin, New York, 1998. Springer.

[213] M. P. Servais, G. de Jager, and J. R. Greene. Function optimisation using multiple-base population based incremental learning. In *Eighth South African Workshop on Pattern Recognition*, Grahamstown, South Africa, 1997. Pattern Recognition Association of South Africa.

[214] I. Servet, L. Travé-Massuyés, and D. Stern. Telephone network traffic overloading diagnosis and evolutionary computation techniques. In J.-K. Hao et. al., editor, *3rd European Conference on Artificial Evolution (AE'97)*, volume 1363 of *Lecture Notes in Computer Science*, pages 137–144, Berlin, New York, 1997. Springer.

[215] Y. Shang and B. W. Wah. Global optimization for neural network training. *IEEE Computer*, 29(3):45–54, 1996.

[216] J. W. Shavlik and R. J. Moody. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6:111–143, 1991.

[217] A. J. Shepherd. *Second-Order Methods for Neural Networks*. Springer-Verlag, London, 1997.

[218] F. M. Silva and L. B. Almeida. Speeding up backpropagation. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 151–158. North-Holland, Amsterdam, Netherlands, 1990.

[219] F. J. Solis and R. J-B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[220] S. A. Solla, E. Levin, and M. Fleisher. Accelerated learning in layered neural networks. *Complex Systems*, 2:625–640, 1988.

[221] S. A. Solla, G. B. Sorkin, and S. R. White. Configuration space analysis for optimization problems. In E. Bienenstock et. al., editor, *Disordered Systems and Biological Organization, NATO ASI Series*, volume F20, pages 283–293, Berlin, New York, 1986. Springer.

[222] E. D. Sontag and H. J. Sussman. Backpropagation can give rise to spurious local minima even for networks without hidden layers. *Complex Systems*, 3:91–106, 1889.

[223] G. B. Sorkin. Efficient simulated annealing on fractal energy landscapes. *Algorithmica*, 6:367–418, 1991.

[224] I. G. Sprinkhuizen-Kuyper and E. J. W. Boers. The error surface of the simplest XOR network has only global minima. *Neural Computation*, 8:1301–1320, 1996.

[225] I. G. Sprinkhuizen-Kuyper and E. J. W. Boers. The error surface of the 2-2-1 XOR network: the finite stationary points. *Neural Networks*, 11(4):683–690, 1998.

[226] P. F. Stadler. Towards a theory of landscapes. Technical Report SFI-TR-95-03-030, Santa-Fe Institute, 1995.

[227] R. Summers and R. Dybowski. Artificial neural networks: from black-box to grey-box modelling. In *16th International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2, pages 1061–1062, New York, 1994. IEEE.

[228] J. Sun, W. I. Grosky, and M. H. Hassoun. A fast algorithm for finding global minima of error functions in layered neural networks. In *International Joint Conference on Neural Networks*, volume I, pages 715–720, New York, 1990. IEEE.

[229] H. J. Sussmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. *Neural Networks*, 5:589–593, 1992.

[230] S. Tamura and M. Tateishi. Capabilities of a four-layered feedforward neural network: Four layers versus three. *IEEE Transactions on Neural Networks*, 8(2):251–255, 1997.

[231] Z. Tang and G. Koehler. Deterministic global optimal FNN training algorithms. *Neural Networks*, 7(2):301–311, 1994.

[232] A. Y. Terekhina. Methods of multidimensional data scaling and visualization (survey). *Automation and Remote Control*, 34(7):1109–1121, 1973.

[233] G. Thimm and E. Fiesler. Neural network initialization. In *International Workshop on Artificial Neural Networks*, volume 930 of *Lecture Notes in Computer Science*, pages 535–542. Springer-Verlag, New York, 1995.

[234] G. Thimm and E. Fiesler. High-order and multilayer perceptron initialization. *IEEE Transactions on Neural Networks*, 8(2):349–359, March 1997.

[235] D. A. Thomas, K. Johnson, and S. Stevenson. Integrated mathematics, science, and technology: An introduction to scientific visualization. *Journal of Computers in Mathematics and Science Teaching*, 15(3):267–294, 1996.

[236] T. Tollenaere. SuperSAB: Fast adaptive backpropagation with good scaling properites. *Neural Networks*, 3:119–122, 1990.

[237] A. Törn and A. Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1989.

[238] N. K. Treadgold and T. D. Gedeon. Simulated annealing and weight decay in adaptive learning: The SARPROP algorithm. *IEEE Transactions on Neural Networks*, 9(4):662–668, 1998.

[239] J. W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, Reading, MA, 1977.

[240] E. van der Poel and I. Cloete. Animating neural network training. *South African Computer Journal*, (7):44–52, 1992.

[241] J. E. Vitela and J. Reifman. Premature saturation in backpropagation networks: mechanism and necessary conditions. *Neural Networks*, 10(4):721–735, 1997.

[242] T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon. Accelerating the convergence of the back-propagation method. *Biological Cybernetics*, 59:257–263, 1988.

[243] G-J. Wang and C-C. Chen. A fast multilayer neural-network training algorithm based on the layer-by-layer optimizing procedures. *IEEE Transactions on Neural Networks*, 7(3):768–775, 1996.

[244] S. Wang and C. Hsu. A self growing learning algorithm for determining the appropriate number of hidden units. In *International Joint Conference on Neural Networks (Singapore)*, pages 1098–1104, New York, 1991. IEEE.

[245] A. S. Weigend and D. E. Rumelhart. The effective dimension of the space of hidden units. In *International Joint Conference on Neural Networks (Singapore)*, pages 2069–2074, New York, 1991. IEEE.

[246] A. S. Weigend and D. E. Rumelhart. Generalization through minimal networks with application to forecasting. In E. Keramidas, editor, *Computing Science and Statistics: 23rd Symposium of the Interface*, pages 362–370, Farifax Station, VA, 1991. Interface Foundation of North America.

[247] E. Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.

[248] J. Wejchert and G. Tesauro. Neural network visualization. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 465–472, San Mateo, CA, 1990. Morgan Kauffman.

[249] J. Wejchert and G. Tesauro. Visualizing processes in neural networks. *IBM Journal of Research and Development*, 35(1/2):244–253, 1991.

[250] O. Wendt and W. König. Cooperative simulated annealing: How much cooperation is enough? Technical Report 97-19, Frankfurt University, Germany, 1997.

[251] D. Whitley and T. Starkweather. Genitor II: A distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, 2(3):189–214, 1990.

[252] B. Widrow and S. D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, Englewood Cliffs, N.J, 1985.

[253] D. J. Wilde and C. S. Beightler. *Foundations of optimization*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[254] P. Wilke. Visualization of neural networks using NEUROGRAPH. *University Education Uses of Visualization in Scientific Computing*, A-48:105–117, 1994.

[255] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, February 1995.

[256] P. M. Wong, T. D. Gedeon, and I. J. Taggart. An improved technique in porosity prediction: A neural network approach. *IEEE Transactions on Geoscience and Remote Sensing*, 33(4):971–980, 1995.

[257] X. Yao. Evolutionary artificial neural networks. In A. Kent, editor, *Encyclopedia of Computer Science and Technology*, volume 33, pages 137–170. Marcel Dekker, New York, 1995.

[258] F. W. Young and P. Rheingans. Visualizing structure in high-dimensional multivariate data. *IBM Journal of Research and Development*, 35(1/2):97–107, 1991.

[259] S. A. Young. *Constructive Neural Network Training Algorithms for Pattern Classification Using Computational Geometry Techniques*. PhD thesis, University of Queensland, Department of Electrical and Computer Engineering, 1996.